Weber
Türschmann

BASIC 1

2. Auflage

UTB

Haupt

Uni-Taschenbücher 588

UTB

Eine Arbeitsgemeinschaft der Verlage

Birkhäuser Verlag Basel und Stuttgart
Wilhelm Fink Verlag München
Gustav Fischer Verlag Stuttgart
Francke Verlag München
Paul Haupt Verlag Bern und Stuttgart
Dr. Alfred Hüthig Verlag Heidelberg
Leske Verlag + Budrich GmbH Opladen
J. C. B. Mohr (Paul Siebeck) Tübingen
C. F. Müller Juristischer Verlag - R. v. Decker's Verlag Heidelberg
Quelle & Meyer Heidelberg
Ernst Reinhardt Verlag München und Basel
F. K. Schattauer Verlag Stuttgart-New York
Ferdinand Schöningh Verlag Paderborn
Dr. Dietrich Steinkopff Verlag Darmstadt
Eugen Ulmer Verlag Stuttgart
Vandenhoeck & Ruprecht in Göttingen und Zürich
Verlag Dokumentation München

Karl Weber
Carl Wolfram Türschmann

BASIC 1

Lehr- und Handbuch der Programmiersprache BASIC
mit wirtschaftswissenschaftlichen
Anwendungsbeispielen

2., überarbeitete und erweiterte Auflage

Verlag Paul Haupt Bern und Stuttgart

Die Autoren:

Karl Weber, Dr. oec. publ., M.S. in Accountancy.
1948–1954 Studium der Volks- und Betriebswirtschaftslehre an der Universität Zürich; 1955 Promotion; 1953–1956 Assistent am Handelswissenschaftlichen Seminar der Universität Zürich; 1956–1963 Wissenschaftlicher Mitarbeiter am Betriebswissenschaftlichen Institut der ETH Zürich; 1959 Studium an der London School of Economics and Political Science; 1963–1964 Studium und Tätigkeit als Instructor of Accountancy an der University of Illinois in Urbana; 1964–1967 Assistant Professor of Accountancy an der University of Illinois; 1966 Habilitation an der Universität Zürich; 1966 Fellow in Accountancy an der University of Illinois; 1967–1969 Associate Professor of Accountancy an der University of Illinois; seit 1969 Professor für Betriebswirtschaftslehre an der Justus-Liebig-Universität Giessen.

Carl Wolfram Türschmann, Dipl.-Ing., Dipl.-Oec, Dr. rer. soc.
1964–1970 Studium des Maschinenbaus an der Technischen Hochschule Darmstadt; 1970–1973 Studium der Wirtschaftswissenschaft an der Justus-Liebig-Universität Giessen; 1970–1972 akademischer Tutor im Fachbereich Wirtschaftswissenschaft der Justus-Liebig-Universität; 1972–1977 wissenschaftlicher Mitarbeiter an der Professur für Betriebswirtschaftslehre V der Justus-Liebig-Universität Giessen; 1973–1976 freier Mitarbeiter an der Abteilung für experimentelle Kardiologie am Max Planck-Institut für physiologische und klinische Forschung (W. G. Kerckhoff-Institut), Bad Nauheim; 1976–1980 Lehrbeauftragter für computergestützte Methoden der empirischen Sozialforschung an der Ruhr-Universität Bochum (Institut für Sportwissenschaft); 1978–1980 Forschungsbeauftragter und Projektleiter des Bundesministeriums für innerdeutsche Beziehungen an der Abteilung für Soziologie der Ruhr-Universität Bochum; 1979 Promotion; seit 1980 Hochschulassistent (Assistenzprofessor) für Betriebswirtschaftslehre an der Justus-Liebig-Universität Giessen.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Weber, Karl:
BASIC: Lehr- u. Handbuch d. Programmiersprache BASIC mit wirtschaftswissenschaftl. Anwendungsbeispielen / Karl Weber; Carl Wolfram Türschmann. – Bern; Stuttgart: Haupt

NE: Türschmann, Carl Wolfram:

1.–2., überarb. u. erw. Aufl. – 1981. –
(Uni-Taschenbücher; 588)
ISBN 3-258-03046-4
NE: GT

Copyright © 1981 by Paul Haupt Berne
Alle Rechte vorbehalten. Jede Art der Vervielfältigung ohne Genehmigung des Verlages ist unzulässig.
Printed in Germany
Satz: IBV Lichtsatz KG, Berlin

Vorwort zur zweiten Auflage

Seit Veröffentlichung der ersten Auflage im Jahr 1977 hat BASIC (Beginner's All-purpose Symbolic Instruction Code) unter allen Programmiersprachen eine dominierende Stellung erreicht. Gefördert wurde diese Entwicklung durch den benutzerfreundlichen Charakter von BASIC im allgemeinen und die fast ausschliessliche Verwendung von BASIC auf Kleincomputersystemen mit Dialogorientierung im besonderen. Hinzu kommt die weitgehende Standardisierung von BASIC durch ANSI (American National Standards Institute) und ECMA (European Computer Manufacturers Association).
Die zweite Auflage der vorliegenden Arbeit wurde unter Mitberücksichtigung der neueren BASIC-Sprachentwicklungen weitgehend überarbeitet; insbesondere wurden die Standards gemäss ANSI/ECMA Minimal BASIC sowie die Standardisierungsvorschläge gemäss ANSI BASIC in den Text integriert.
Die ANSI Standards wurden auch in eine weiterentwickelte Version des FOSBIC-Compilersystems eingebaut; darauf wird im vorliegenden Text unter spezieller Bezugnahme auf die FOSBIC-Version 2 hingewiesen. Im übrigen betreffen die Angaben zum FOSBIC-Compiler stets die Version 1, die in

> Karl Weber und Carl Wolfram Türschmann. FOSBIC-Compiler. Ein BASIC-Compilersystem auf FORTRAN-Basis.
> Bern 1977

veröffentlicht wurde.
Die Version 1 des FOSBIC-Compilersystems wurde seit 1977 von verschiedenen Instituten und Unternehmungen übernommen; erwähnt seien:

> Honeywell Bull GmbH, Eschbach/Ts.
> IFNS Institut für Numerische Statistik, Köln
> K. Marx University of Economics, Department of Mathematics, Budapest, Ungarn
> Regionales Rechenzentrum Erlangen, Erlangen
> University of Lodz, Institute of Econometrics and Statistics, Lodz, Polen
> University of North Carolina at Wilmington, Department of Business and Economics, Wilmington, North Carolina, USA

Die FOSBIC-Version 2 ist zur Zeit an der Justus-Liebig-Universität Giessen.

im praktischen Einsatz und wird insbesondere im Rahmen der BASIC-Programmierkurse des Fachbereichs Wirtschaftswissenschaft getestet. Es bleibt weiterhin Aufgabe der Autoren, die Entwicklung von BASIC zu erfassen und in standardisierter Form in FOSBIC-BASIC zu integrieren. Nach Erreichen von in sich abgeschlossenen Entwicklungseinheiten wird das jeweils neueste FOSBIC-Compilersystem freigegeben und veröffentlicht.

Karl Weber
Carl Wolfram Türschmann Giessen 1981

Vorwort zur ersten Auflage

Die rechnerische Behandlung komplexer Probleme erfolgt in allen Wissenschaftsgebieten in zunehmenden Masse unter Verwendung elektronischer Rechenanlagen, zu deren Programmierung an Hochschulen bisher zumeist FORTRAN, COBOL oder ALGOL verwendet wurden. Die Erlernung dieser Programmiersprachen erfordert erfahrungsgemäss einen erheblichen Zeitaufwand, so dass die EDV-orientierte Behandlung komplexer Probleme in der Regel erst in höheren Semestern erfolgen kann.

Im Gegensatz zu den vorerwähnten Programmiersprachen erweist sich BASIC als relativ leicht erlernbar und damit bereits im Grundstudium voll verwendbar. BASIC kommt damit tatsächlich die Funktion eines „**B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode" zu und vermochte sich aus diesem Grunde an amerikanischen Hochschulen – insbesondere als Dialogsprache – in weitgehendem Masse durchzusetzen. Dagegen fand BASIC an europäischen Hochschulen bisher vergleichsweise geringe Verbreitung, was teilweise auf das Fehlen eines auch für den Stapelbetrieb geeigneten und auf bestehende Grossrechenanlagen übernehmbaren BASIC-Compiler zurückzuführen ist.
Mit der Publikation des FOSBIC-Compilersystem in

> Karl Weber und Carl Wolfram Türschmann
> FOSBIC-Compiler. Ein BASIC-Compilersystem auf
> FORTRAN-Basis. Bern und Stuttgart 1977

wird der letzterwähnte Mangel insofern beseitigt, als nunmehr ein hochkompatibler BASIC-Compiler mit relativ geringen Speicherplatzanforderungen allgemein verfügbar ist.
Um die rasche Verbreitung von FOSBIC-BASIC zu fördern, wird im vorliegenden Lehr- und Handbuch neben anderen speziell auf diese BASIC-Variante Bezug genommen. Dabei ist zu erwähnen, dass FOSBIC-BASIC – im Vergleich zu anderen BASIC-Dialekten – in weitestgehendem Masse den ANSI/ECMA BASIC-Standards gerecht wird. Diese Standards wurden bei der Behandlung der BASIC-Elementaranweisungen und -Sonderanweisungen angemessen in den Text integriert.
Weiterhin sind für Wirtschaftswissenschaftler relevante BASIC-Programmbeispiele in den Text eingefügt. Es handelt sich hierbei zumeist um neuentwickelte Programme; in einem Fall ist ein BASIC-Anwenderpro-

gramm in überarbeiteter Form aus den Dartmouth Public Files übernommen.
Die Verfasser haben sich zur Aufgabe gestellt, für den deutschen Sprachbereich als Informationsstelle zur BASIC-Standardisierung und als Austauschstelle für BASIC-Programme zu dienen.

Kontaktadresse: Professur für Betriebswirtschaftslehre V, Justus Liebig-Universität Giessen, Licher Strasse 74, D-6300 Giessen.

Karl Weber
Carl Wolfram Türschmann

Giessen 1977

Inhaltsverzeichnis

Einleitung 13
1 BASIC-Charakteristik 14
11 BASIC Dialogbetrieb 14
111 Version des Dartmouth College 15
112 Weiterentwicklungen 20
112.1 General Electric Company 20
112.2 Honeywell Bull Corporation 21
112.3 IBM Corporation 22
112.4 Philips-Electrologica B. V. 23
112.5 Computer Gesellschaft Konstanz mbH 24
112.6 National Cash Register Company 24
112.7 International Computer Limited 25
112.8 BASIC Timesharing, Inc. 25
112.9 Com-Share Incorporated 26
12 BASIC Monologbetrieb 27
121 Version der University of Washington 27
122 Weiterentwicklungen 29
122.1 Worcester Polytechnic Institute 29
122.2 Justus-Liebig-Universität Giessen 30
13 BASIC-Alternativbetrieb 32
131 Version der University of Maryland 33
132 Weiterentwicklungen 33
132.1 Sperry Rand Corporation 33
132.2 Burroughs Corporation 34
132.3 CII Honeywell Bull 34
132.4 Control Data Corporation 35
132.5 Siemens Aktiengesellschaft 35
132.6 Xerox Data System 36
132.7 International Timesharing Corporation 36
132.8 Justus-Liebig-Universität Giessen 38
14 BASIC-Standardisierung 42
141 ANSI-, ECMA- und ISO-Standards 42
142 Weiterentwicklungen 46
2 BASIC-Problembearbeitung 47
21 Problembeschreibung und Problemanalyse 47
22 Programmplanung 48
221 Datenflusspläne 48
222 Programmablaufpläne 49
23 Programmierung 58
231 Codierung 58

232 Programmübertragung 58
232.1 Monologbetrieb 58
232.2 Dialogbetrieb 59
233 Programmumwandlung 60
233.1 Monologbetrieb 61
233.2 Dialogbetrieb 61
234 Programmtest 62
24 Programmdokumentation 62
25 BASIC-Sprachausschöpfung 63
3 BASIC-Struktur 65
31 Formalisierte Sprachdarstellung 65
32 Definitionen 66
321 Grundsymbole 66
321.1 Alphazeichen 68
321.2 Ziffern 69
321.3 Sonderzeichen 70
322 Konstante 75
322.1 Numerische Konstante 76
322.11 Ganzzahlen 76
322.12 Gleitkommazahlen 80
322.2 Alphanumerische Konstante 81
323 Variable 83
323.1 Numerische Variable 84
323.11 Einfache numerische Variable 85
323.12 Indizierte numerische Variable 86
323.2 Alphanumerische Variable 89
323.21 Einfache alphanumerische Variable . . 89
323.22 Indizierte alphanumerische Variable . . 91
323.23 Teilzeichenbegrenzung alphanumerischer Variabler 92
324 Ausdrücke 94
324.1 Arithmetische Ausdrücke 96
324.2 Alphanumerische Ausdrücke 101
324.3 Logisch-komparative Ausdrücke 102
324.31 Logische Ausdrücke 102
324.32 Komparative Ausdrücke 104
325 Funktionen 106
325.1 Numerische Funktionen 106
325.11 Eingebaute numerische Funktionen . . 107
325.111 Elementarfunktionen 107
325.112 Zusatzfunktionen 110
325.12 Benutzerdefinierte num. Funktionen . . 112
325.2 Alphanumerische Funktionen 114

325.21 Eingebaute alphanum. Funktionen . . . 114
325.22 Benutzerdefinierte alphanumerische Funktionen 115
325.3 Hybridfunktionen 116
325.31 Numerische Hybridfunktionen 116
325.311 Eingebaute numerische Hybridfunktionen 116
325.312 Benutzerdefinierte numerische Hybridfunktionen 118
325.32 Alphanumerische Hybridfunktionen . . 118
325.321 Eingebaute alphanumerische Hybridfunktionen 119
325.322 Benutzerdefinierte alphanumerische Hybridfunktionen 120
326 Anweisungen 122
326.1 Statementstruktur 123
326.2 Statementnumerierung 125
326.3 Codierblätter 128
33 Elementaranweisungen 129
331 Nichtausführbare Anweisungen 130
331.1 Kommentare 130
331.11 REM-Statement 131
331.12 null-Statement 131
331.13 COMMENT-Statement 132
331.2 Vereinbarungen 133
331.21 DIM-Statements 134
331.211 Numerische DIM-Statements . 135
331.212 Alphanumerische DIM-Statements 137
331.213 Allgemeine DIM-Statements . 139
331.22 BASE-Statement 140
331.23 DEF-Statements 147
331.231 Numerische DEF-Statements . 147
331.232 Alphanumerische DEF-Statements 152
331.233 Hybride DEF-Statements . . 154
332 Ausführbare Anweisungen 157
332.1 Eingabeanweisungen 158
332.11 DATA-Statement 159
332.12 data-Statement 162
332.13 READ-Statement 163
332.14 INPUT-Statement 166
332.15 RESTORE-Statements 169

332.2 Ausgabeanweisungen 173
332.21 PRINT-Statement 174
332.22 PRINT TAB-Statement 183
332.23 PRINT USING- und IMAGE-Statement 185
332.24 Allgemeines PRINT-Statement 194
332.25 PAGE-Statement 195
332.3 Wertzuordnungsanweisungen 195
332.31 LET-Statements 196
332.311 Numerisches LET-Statement 198
332.312 Alphanumerisches LET-Statement 200
332.32 let-Statements 204
332.4 Programmsteuerungsanweisungen 205
332.41 GOTO-Statements 206
332.411 Unbedingte GOTO-Statements 207
332.412 Bedingte GOTO-Statements . 211
332.42 IF-Statements 216
332.421 Einfache IF-Statements . . . 217
332.422 IF-Block 223
332.43 FOR/NEXT-Statements 224
332.44 LOOP/EXIT-Statements 233
332.45 STOP-Statements 234
332.46 END-Statements 235
332.47 GOSUB-Statements 238
332.471 Unbedingtes GOSUB-Statement 238
332.472 Bedingte GOSUB-Statements 239
332.48 RETURN-Statement 241
332.49 Unterprogrammtechnik mit GOSUB und RETURN 242
34 Programmaufbau 244
4 BASIC-Anwendungsbeispiele 246
41 Betriebswirtschaftslehre 246
411 Exponentielle Glättung 246
412 Critical Path Method (CPM) 259
42 Volkswirtschaftslehre 254
421 Preiselastizität der Nachfrage 254
422 Oszillationsmodell nach J. R. Hicks 259
43 Statistik 266
431 Statistische Masszahlen 266
432 Lineare Regression und Korrelation 270

Einleitung

Das vorliegende Lehr- und Handbuch soll als BASIC-Einführung und zur BASIC-Vertiefung dienen.
Band 1 orientiert in einem ersten Kapitel über die BASIC-Entwicklung im allgemeinen und vermittelt einen Überblick über eine Reihe von Maschinensystemen, die BASIC als Programmiersprache zulassen. Das zweite Kapitel führt in die Grundlagen der BASIC-Programmierung ein. Im dritten Kapitel wird BASIC unter besonderer Berücksichtigung der BASIC-Elementaranweisungen behandelt. Der metalinguistischen Darstellungsweise von Backus folgend, wird die BASIC-Syntax dargestellt und verbal die BASIC-Semantik erörtert. Zur Illustration der theoretischen Ausführungen dienen zahlreiche Beispiele, die mit dem – im Vorwort erwähnten – FOSBIC-Compiler bearbeitet wurden. Das vierte Kapitel vermittelt einige praxisnahe, speziell für Wirtschaftswissenschaftler relevante Beispielprogramme.
BASIC-Anfänger sollten sich zunächst intensiv um die Durcharbeitung des dritten Kapitels und der Beispielprogramme bemühen. Bei Verständnisschwierigkeiten im logischen Ablauf komplexer Programme empfiehlt sich die Erarbeitung des zweiten Kapitels.
Band 2 befasst sich im ersten Kapitel mit den BASIC-Programmiersystemen, unter besonderer Berücksichtigung der beginnenden Standardisierung von BASIC-Systembefehlen; ausserdem werden im ersten Kapitel BASIC-Minicomputersysteme dargestellt. Im zweiten Kapitel wird die BASIC-Problembearbeitung an einem Beispiel gezeigt. Das dritte Kapitel befasst sich mit den BASIC-Sonderanweisungen, wobei auch Besonderheiten des FOSBIC-Compilers verdeutlicht werden. Im vierten Kapitel werden – analog zu Band 1 – praxisbezogene Beispielprogramme aus dem Gebiet der Wirtschaftswissenschaften behandelt. Im fünften Kapitel wird die – zumeist angloamerikanische – BASIC-Literatur besprochen. Auch in Band 2 kann zunächst auf die Durcharbeitung der beiden ersten Kapitel verzichtet und mit dem Studium des dritten Kapitels begonnen werden. Weiterhin bietet sich für Wirtschaftswissenschaftler die Übernahme einzelner Programme aus dem vierten Kapitel an.

1 BASIC-Charakteristik

BASIC (**B**eginner's **A**ll-Purpose **S**ymbolic **I**nstruction **C**ode) ist eine benutzerorientierte Programmiersprache, die anfangs der sechziger Jahre in der Absicht entwickelt wurde, Hochschulstudenten möglichst rasch und effizient zu selbständigen Benutzern von Datenverarbeitungsanlagen auszubilden. Dementsprechend wurde BASIC primär als einfach zu erlernende und formal leicht verständliche Programmiersprache konzipiert, die dem Benutzer späterhin auch den Zugang zu anderen Programmiersprachen, insbesondere FORTRAN und ALGOL, erleichtern sollte.[1]

BASIC ist eine lernorientierte Programmiersprache.

BASIC wurde – beeinflusst durch die benutzerfreundlichen Programmiersprachen JOSS und CORC[2] – am Dartmouth College, Hanover, speziell für den Dialogbetrieb[3] entwickelt und gilt als eine für die direkte Datenfernverarbeitung besonders geeignete Programmiersprache.
BASIC wird aber nicht nur im Dialogbetrieb verwendet, sondern – im Anschluss an entsprechende Pionierarbeiten der Universitäten Maryland und Washington – auch im Alternativ- und Monologbetrieb eingesetzt.

11 BASIC-Dialogbetrieb

BASIC vermochte sich – in den USA – als für Datenfernverarbeitung besonders geeignete Programmiersprache, insbesondere auch gegenüber

1 "BASIC was meant to be a very simple language to learn and also one that would be easy to translate. Furthermore, the designers wished it to be a stepping-stone for students to learn one of the more powerful languages such as FORTRAN or ALGOL." Jean E. Sammet. Programming Languages: History and Fundamentals. Englewood Cliffs 1969, p. 229.
2 Zu diesen Programmiersprachen vgl. Sammet, Programming Languages, p. 217–226 und p. 294–296.
3 Vgl. Roland Kraushaar, Leo Jakob und Dieter Goth. Datenfernverarbeitung. 2. Aufl., München 1974.

APL[4], weitgehend durchzusetzen: „Despite the rapid growth of APL, BASIC is by far the most widely-used interactive language."[5]

BASIC ist die meistverwendete Dialogsprache.

Die Entwicklung von Dialog-BASIC führte nicht nur zum Einsatz entsprechender Sprachprozessoren (Interpretern), sondern auch zum Aufbau eigenständiger BASIC-Programmiersysteme. Sie ermöglichen die Speicherung neu entwickelter BASIC-Programme in einer Programmbibliothek, die Übernahme von BASIC-Programmen aus allgemein verfügbaren (öffentlichen) oder benutzereigenen (privaten) Programmbibliotheken sowie den Ersatz bzw. die Elimination von Bibliotheksprogrammen. Vgl. Abbildung 1. Einzelheiten über die BASIC-Programmiersysteme sind aus Band 2, Abs. 11 ersichtlich.

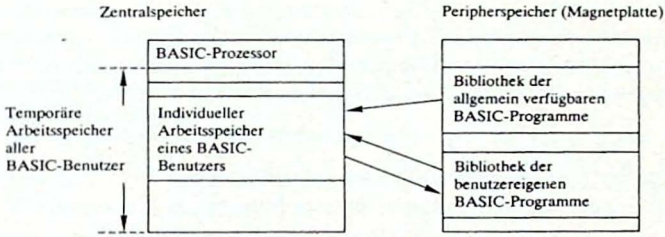


Abbildung 1. BASIC-Programmiersystem

111 Version des Dartmouth College

Die ursprüngliche Version von BASIC wurde – mit Unterstützung der amerikanischen National Science Foundation – 1963–64 am Dartmouth College unter der Leitung von John G. Kemeny und Thomas E. Kurtz als Dialogsprache für das Time-Sharing Computersystem GE-265 entwikkelt.[6]

BASIC wurde am Dartmouth College als Dialogsprache konzipiert.

4 Über APL vgl. Sammet, Programming Languages, p. 247–253.
5 Progress in Interactive Computing. Computing Newsletter for Schools of Business 7 (1974) 7, March, p. 1. Vgl. auch Bennet P. Lientz. A Comparative Evaluation of Versions of BASIC. Communications of the ACM 19 (1976) 4, April, p. 175.
6 J. Daniel Couger. Computers and the Schools of Business. Boulder 1967, p. 68–75.

Dartmouth BASIC-1 war in höchstem Masse auf die Bedürfnisse von Programmier-Anfängern ausgerichtet[7] und wies deshalb eine sich im wesentlichen selbst erklärende Statement- und Programmstruktur auf. Weiterhin zeichnete sich Dartmouth BASIC-1 durch enge Verbindungen zu FORTRAN und ALGOL aus, so dass BASIC-Benutzern der Einstieg in diese höheren Programmiersprachen und eventuelle Programmkonversionen besonders leicht fallen sollten.[8]

BASIC weist im Vergleich zu FORTRAN insbesondere dadurch eine vereinfachte Struktur auf, als auf die Definition von zwei Variablentypen (real, integer), auf Formatstatements und dreifache IF-Verzweigungen verzichtet wurde. Weiterhin wurde bei BASIC von der Möglichkeit zur impliziten Programmierung (default convention) in besonderem Masse Gebrauch gemacht, beispielsweise bei der Dimensionierung von Vektoren (0:10) und Matrizen (0:10, 0:10). Dagegen ergab sich für BASIC als Dialogsprache die Notwendigkeit zur durchlaufenden Numerierung aller Statements.

Die besondere Struktur der Programmiersprache BASIC kommt im folgenden Beispiel recht deutlich zum Ausdruck; es handelt sich hierbei um ein Programm zur Lösung einer Gleichung zweiten Grades, $ax^2 + bx + c = 0$, a ungleich Null. Für sie gilt generell $x_{1,2} = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$. Die Lösungen x_1, x_2 werden reell und ungleich, wenn der Wert unter der Wurzel positiv ist, beziehungsweise gleich, wenn die Diskriminante Null ist. Konjugiert komplexe Lösungen entstehen, wenn der Wert unter der Wurzel negativ ist[9]. Falls a gleich 0 ist, ergibt sich aus der linearen Beziehung $bx + c = 0$ nur die Lösung $x = -c/b$.

7 "The original version of BASIC was designed to be a purely beginner's language." Stephen J. Garland. Dartmouth BASIC: A Specification. Hanover 1973, p. 3.

8 "Every effort was made to design each statement and each feature with the novice in mind. Put another way, most BASIC statements were designed so that their purpose was self-evident. ... Another design criterion was to avoid severe departures from existing programming languages. Thus the conversion of most programs in FORTRAN or ALGOL to equivalent programs in BASIC, or vice versa, is fairly straightforward, being made almost on a line for line basis." Garland, Dartmouth BASIC, p. 2.

9 Vgl. G. Arnold. Formeln der Mathematik. Braunschweig 1965, S. 52. – Das vorerwähnte Beispiel wird – in etwas vereinfachter Form – in der BASIC-Literatur oft verwendet. Compagnie International pour l'Informatique. 10070 BASIC User's Manual, Paris 1969, p. 1–2; Xerox Data Systems. Xerox BASIC Sigma 5–9 Computers. Language and Operations Reference Manual. El Segundo 1971, p. 1–2; Jacques Alazard et Jean-Pierre Lamoitier. Le langage BASIC et ses extensions. Paris 1973, p. 41.

```
 10  REM GLEICHUNG ZWEITEN GRADES
 20  PRINT ≠          A≠,≠          B≠,≠          C≠,≠          X(1)≠,≠          X(2)≠,
 25  PRINT ≠ BEMERKUNG≠
 30  READ A,B,C
 40  IF A <> 0 THEN  80
 50  IF B <> 0 THEN 140
 60  IF C <> 0 THEN 190
 70  GOTO 900
 80  LET D = B**2-4*A*C
 90  IF D < 0 THEN 170
100  LET X1=(-B+SQR(D))/(2*A)
110  LET X2=(-B-SQR(D))/(2*A)
120  PRINT A,B,C,X1,X2,≠ REELL≠
130  GOTO 30
140  LET X=-C/B
150  PRINT A,B,C,X,≠ ≠,≠ LINEAR≠
160  GOTO 30
170  PRINT A,B,C,≠ ≠,≠ ≠,≠ KOMPLEX≠
180  GOTO 30
190  PRINT ≠FALSCHE DATENEINGABE≠
200  GOTO 30
210  DATA -1, 6.2, 3, 1, 1, 1, 0, 0, 0
220  REM  SIEHE ABBILDUNG 11
900  STOP
950  END
```

Im vorstehenden Programm werden neun verschiedene BASIC-Statements verwendet, denen die Funktion von

- Kommentaren (REM),
- Eingabeanweisungen (READ, DATA),
- Wertzuordnungsanweisungen (LET),
- Ausgabeanweisungen (PRINT),
- Programmsteuerungsanweisungen (GOTO, IF/THEN, STOP, END) zukommt.

Der Dartmouth BASIC-Compiler bot in seiner Ursprungsversion zusätzlich noch die Möglichkeit zur

- Dateneingabe mit INPUT,
- Wertzuordnung ohne LET,
- Verwendung eingebauter Funktionen,
- Abarbeitung von Schleifen und Unterprogrammen mit FOR/NEXT und GOSUB/RETURN,
- Matrizenverarbeitung mit verschiedenen MAT-Statements.[10]

Dartmouth BASIC-1 enthielt damit bereits alle wesentlichen Eigenschaften einer höheren Programmiersprache.

10 "The first issue of the programming manual from Dartmouth College... was in January 1965. This version included those statements which may be regarded as the core of the system – let, read, data, print, goto, if, for, next, end, single-line function definitions, gosub, return, remark, the mat statements and stop. Very shortly after this publication, the input statement was added." G. M. Bull, W. Freeman and S. J. Garland. Specification for Standard BASIC. Manchester 1973, p. 9.

Seit 1965 wurde Dartmouth BASIC wesentlich erweitert.[11] Hierzu bot teilweise der Übergang zu leistungsfähigeren Datenverarbeitungssystemen Anlass;[12] insbesondere kam es aber deshalb zu Weiterentwicklungen – etwa durch Einbezug spezieller Matrizenanweisungen, Fileanweisungen usw. –, weil BASIC nicht im anfänglich erwarteten Masse als Ausgangsbasis zur Erlernung von FORTRAN und ALGOL, sondern vielmehr als selbständige Sprache zur Programmierung komplexer Probleme Verwendung fand.[13]
Bei allen Spracherweiterungen wurden die Grundkonzepte von BASIC – leichte Verständlichkeit und Erlernbarkeit – beibehalten.[14] Weiterhin

11 "By January 1966, the third edition of the programming manual was published. The system had now been extended to include the restore statement, a few more functions and a few more commands. By March 1967 the system included the randomize statement, tabulation and string variables. By July 1967 multiline function definitions were included as were random access and terminal format files (this included write, reset, scratch and file statements), chaining and segmentation using gosub channel number and return. By February 1969 mat input/output to files had been added as had the linput, margin and change statements, file conditions and a number of standard string functions. By September 1970 linput and mat-linput from files had been added and the manual was also the most comprehensive so far produced. In October 1970 decisions were made regarding new subprogram facilities, a revised file statement, debugging facilities and improved string handling." Bull, Freeman and Garland, Specification for Standard BASIC, p. 10.

12 1967 wurde das Timesharing Computersystem GE-265 installiert und damit die Möglichkeit zum Anschluss von 200 Datenstationen geschaffen. Weitere Entwicklungsarbeiten führten – unter Einbezug des Computersystems Honeywell H 635 – zum Dartmouth Time-Sharing System (DTSS). Honeywell Information Systems. Series 600/6000 Software. Introduction to DTSS Software. Waltham 1973, p. 12–13; Honeywell Information Systems. Series 600/6000 Software. Dartmouth Time-Sharing System (DTSS). Waltham 1973.

13 "[Originally,] it was expected that persons requiring more sophisticated programming techniques would graduate to a general-purpose language such as ALGOL or FORTRAN. When it was realized that human nature made this plan impractical, new features and statements were added to BASIC." Garland, Dartmouth BASIC, p. 3.

14 "In summary, the following are the major criteria employed in the design of new features in BASIC. (1) The action of features should be self-evident. (2) Features should be easy to learn. (3) Users should not have to learn about a feature before they have a specific need for that feature. (4) Programming details should be handled wherever possible by default conventions and not be brought to the novice's attention; experienced users should be able to override the defaults. (5) Features should be designed with the needs of the user taking priority over the needs of the implementation. (6) Simple tasks should have simple solutions." Garland, Dartmouth BASIC, p. 4.

sollte, den Bedürfnissen der Benutzer entsprechend, alternativ mit verschiedenen BASIC-Versionen gearbeitet werden können. Diese Ziele liessen sich durch einen modularen Aufbau des Dartmouth BASIC-Compilers erreichen. Im wesentlichen sind folgende BASIC-Module verfügbar:[15]

- Minimal BASIC, im wesentlichen Dartmouth BASIC-1 entsprechend;
- Extended BASIC, mit grösserer Zahl von Standardfunktionen (COT, MOD, NUM, CLK$, DAT$ usw.), mehrzeiligen Benutzerfunktionen, ON/GOSUB-Statement usw.;
- String Package, speziell zur Verarbeitung alphanumerischer Variablen;
- Matrix Package, mit Spezialanweisungen zur Verarbeitung von Vektoren und Matrizen, zusätzlichen Standardfunktionen (DET);
- Sequential File Package, zur Verarbeitung sequentiell organisierter Dateien;
- Random Access File Package, zur Verarbeitung index-sequentieller Dateien;
- Subprograms, mit SUB und CALL-Statements.

Dartmouth BASIC ist in verschiedenen modular aufgebauten Versionen verfügbar.

Vermerkt sei auch, dass am Kiewit Computation Center spezielle FORTRAN/BASIC-Übersetzer (translators) entwickelt wurden, so dass die unter amerikanischen Hochschulen – im Rahmen des Projekts CONDUIT – austauschbaren Softwarepakete sowohl in ANSI Standard FORTRAN als auch in Dartmouth BASIC verwendbar sind.[16]
Bezüglich des wirtschaftswissenschaftlichen Fachbereichs des Dartmouth College sei erwähnt, dass die Amos Tuck School of Business Administration seit Einführung des Time-Sharing Systems über eigene Datenendsta-

15 Garland, Dartmouth BASIC, p. 85–86. Die Dartmouth BASIC-Module wurden durch ANSI BASIC weitgehend normiert; dadurch wurde eine erhebliche Erweiterung von ANSI Minimal BASIC erreicht.

16 CONDUIT ist eine Kurzform für *C*omputers at *O*regon State University, *N*orth Carolina Educational Computing Service, *D*artmouth College and the *U*niversities of *I*owa and *T*exas. Assoziierte Mitglieder sind Indiana University, Northwestern University und University of Colorado. – Dartmouth College ist im Rahmen des CONDUIT Projekts für Software aus dem Gebiet der Physik verantwortlich. Vgl. CONDUIT Pipeline 1 (1974) 1, p. 4 und 1 (1974) 2, September, p. 4.

tionen verfügt. Die Programmiersprache BASIC wird in Lehre und Forschung verwendet, wobei auch auf Entwicklung und Unterhalt leistungsfähiger BASIC-Softwarepakete besonderer Wert gelegt wird.[17]

112 Weiterentwicklungen

112.1 General Electric Company

Das in Zusammenarbeit mit dem Dartmouth College realisierte (erste) Time-Sharing System war ein grosser Erfolg, so dass General Electric 1965 mit dem Aufbau eines unternehmungseigenen Time-sharing Service-Netzes begann. Damit konnten leistungsfähige Computersysteme auch Unternehmungen zugänglich gemacht werden, die über keine eigene Anlage verfügten und denen die Datenverarbeitung in Service-Bureaux als zu teuer oder zu umständlich erschien.

Die im GE Time-Sharing System bevorzugt angewandte Programmiersprache war BASIC.[18] Als leicht zu erlernende Sprache ermöglichte sie dem Benutzer einen effizienten Direktverkehr mit dem Computer, ohne Zwischenschaltung von Programmierern, und damit eine volle Ausnutzung des Time-Sharing-Effektes.[19]

Unter Berücksichtigung der Weiterentwicklungen von Dartmouth BASIC wurde auch GE BASIC sukzessive ausgeweitet, insbesondere durch Übernahme von PRINT USING-Statement, zusätzlicher Funktionen (RND, DEF) und durch Ausbau der Fileverarbeitung (BCD/Binary Data Files).[20]

Basierend auf unternehmungsinternen GE-Kursen erschienen ab 1966 die ersten BASIC-gestützten Lehrbücher zur Management-Ausbildung. Sie waren geeignet, BASIC als benutzerfreundliche, für das wirtschaftswissenschaftliche Studium besonders geeignete Programmiersprache zu etablieren.[21]

17 Vgl. etwa: J. Peter Williamson and David H. Downes. Manuals for Computer Programs in Finance and Investments. 3rd ed., Hanover 1973.

18 General Electric. GE-400 Series. BASIC Language for Time Sharing Programming System. Reference Manual. Phoenix 1968.

19 Vgl. hierzu: William T. Newell und Susie Fischli. Entscheidungsvorbereitung und Time-Sharing. Industrielle Organisation 39 (1970) 7, S. 299–304.

20 General Electric. BASIC Language. Reference Manual. Rockville 1974; General Electric. International BASIC 1 Time Sharing Manual. Bethesda 1971.

21 Vgl. Clifford H. Springer, Robert E. Herlihy, Robert T. Mall and Robert I. Beggs. Statistical, Inference. Homewood 1966; Clifford H. Springer, Robert E. Herlihy, Robert T. Mall and Robert I. Beggs. Probabilistic Models. Homewood 1968.

BASIC ist eine für das Studium der Wirtschaftswissenschaften relevante Programmiersprache.

112.2 Honeywell Bull Corporation

Der – hier nicht in Einzelheiten zu besprechende – Übergang der GE Computer-Abteilung an die Honeywell-Gruppe führte in Europa auch zur Übernahme des GE Time-Sharing Service durch die Compagnie Honeywell Bull (Paris[22]) und deren Tochtergesellschaften (in der Bundesrepublik Deutschland Honeywell Bull GmbH in Köln).[23] Eingesetzt werden die GE-Systeme MARK I und MARK III, wobei das letztere auch die Möglichkeit zum Stapelbetrieb bietet.[24] Vgl. auch hinten, Abs. 132.3. Benutzer des HB Time-Sharing Systems können grundsätzlich mit den Programmiersprachen ABASIC, BASIC, ALGOL und FORTRAN arbeiten. Besonders empfohlen wird die Verwendung von BASIC, insbesondere im Hinblick auf die Möglichkeit zur String- und Dateiverarbeitung sowie zur formatierten Ausgabe. Demgegenüber läßt ABASIC die Verarbeitung von Magnetplattendateien und alphabetischer Daten nicht zu, bietet dafür aber höhere Kompilationsgeschwindigkeiten.[25]
Die am weitesten entwickelten HB BASIC-Versionen sind über Computersysteme der Serien 600/6000 und 700 verfügbar.[26] Der HB Time-Sharing Service zeichnet sich durch vielseitig ausgebaute Programmbibliotheken aus. Die allen Teilnehmern zur Verfügung stehenden wirtschaftswissenschaftlich orientierten Programme betreffen beispiels-

22 An der Compagnie Honeywell Bull sind die Firmen Honeywell Information Systems Inc. (66%) und die Compagnie des Machines Bull (34%) kapitalmässig beteiligt, wobei die Honeywell Bull Information Systems Inc. ihrerseits eine Gemeinschaftsunternehmung der Firmen Honeywell Inc. (81,5%) und General Electric Company (18,5%) ist.

23 Honeywell Bull. Informations-Service MARK I/MARK III. Time-Sharing BASIC Handbuch. Köln 1974; Honeywell Bull. Time-Sharing Service MARK I. Time-Sharing BASIC Handbuch. 3. Aufl., Köln 1972.

24 General Electric. BASIC Language. MARK III Foreground Reference Manual. Bethesda 1973; Honeywell Bull. Time-Sharing Service MARK III. BASIC Language. Köln 1972; Honeywell Bull. Time-Sharing Service MARK III. MARK III Taschenlexikon. Köln 1973.

25 Honeywell Bull. Time-Sharing Service: Systemübersicht. Köln 1972. S. B–5.

26 Honeywell. Series 600/6000 Software. Time-Sharing BASIC. Waltham 1973. Honeywell. Series 600/6000 dataBASIC System. Language Manual. Wellesley Hills 1971; Honeywell. System 700. BASIC OS/700. Waltham 1972.

weise auch Tilgungspläne bei gleichbleibender und abnehmender Annuität, Cash-Flow Analysen, Wertpapier-Analysen, Lineare Programmierung und lineare Regression.[27]
Der HB Time-Sharing Service wurde in Europa frühzeitig von wissenschaftlichen Forschungsinstituten und bekannten Grossunternehmungen benutzt. Verwiesen sei insbesondere auf die Praxisberichte von Esso Europa Inc., London, und AEG-Telefunken, Frankfurt.[28]

BASIC ist eine in der betriebswirtschaftlichen Forschung und Praxis erfolgreich eingesetzte Programmiersprache.

112.3 IBM Corporation

Die am Dartmouth College entwickelte Dialogsprache BASIC wurde 1968 auch von IBM übernommen und in der Folge weiter ausgebaut.[29]
IBM BASIC umfasst eine grössere Zahl eingebauter Funktionen (intrinsic functions), vordefinierte Konstante (internal constants), bietet insbesondere auch die Möglichkeit zum Rechnen mit einfacher und doppelter Genauigkeit (BASIC, BASICL) und zur externen Datenspeicherung. Hervorzuheben sind auch die Systembefehle zur Erleichterung der Programmkorrektur (debugging aids).[30]
Die wichtigsten IBM-Implementierungen von BASIC sind:
- System 3/BASIC für Benutzer von IBM-System/3 Modell 6.[31]
- ITF BASIC für System 360 OS/DOS. ITF (Interactive Terminal Facility) bildet auch die Grundlage zum Teilnehmerbetrieb auf System/370 und wurde in erster Linie für Problemprogrammierer entwickelt.

27 Vgl. Honeywell Bull. Time-Sharing Service MARK I. Programm-Bibliothek. 3. Aufl., Köln 1972; Honeywell Bull. Time-Sharing Service MARK II. Programm-Bibliothek. 1. Aufl., Köln 1972.
28 Bull General Electric. Der B-GE Time-Sharing Service im Urteil seiner Benutzer. Köln 1970.
29 Ulrich E. Fischer. BASIC – Eine Dialogsprache. IBM-Nachrichten 23 (1973) 214, Februar, S. 549–555.
30 IBM. BASIC Language Reference Manual. New York 1970.
31 IBM. System/3 Model 6. Guide to BASIC. Learner-Paced Format. 2nd ed., New York 1971; IBM. IBM System/3 Model 6. System/3 BASIC Reference Manual. 4th ed., New York 1972 = IBM. IBM System/3 Modell 6. System/3 BASIC Handbuch. Stuttgart 1973.

- VS BASIC für System/370.[32]
- CALL: BASIC im Rahmen des CALL-Teilnehmersystems einer Reihe von IBM-Servicezentren.[33]

Weiterhin wurden von IBM auch benutzerorientierte Programmpakete entwickelt, die im Rahmen des Time-Sharing-Betriebs verfügbar sind. Besonders hervorzuheben ist Business Analysis/BASIC, ein aus dreissig Routinen bestehendes Programmsystem.[34]
Von IBM wird BASIC auch zum Informatikunterricht in Schulen empfohlen, wozu auch reichhaltiges Instruktionsmaterial verfügbar ist.[35]

BASIC ist zum Informatikunterricht in Schulen geeignet.

112.4 Philips-Electrologica B.V.

BASIC Compiler sind für die Philips Computer P9200 und P800M verfügbar.[36]
Das Philips P9200 Time-Sharing-System weist eine analoge Grundstruktur auf, wie das GE-265 Time-Sharing Computersystem. Bis zu 32 Benutzer können über ihre Terminals mit dem System gleichzeitig arbeiten und es sind bis zu 245 Benutzer anschliessbar.
P9200 BASIC deckt sich im wesentlichen mit Dartmouth BASIC-1. Als zusätzliche Standardfunktionen erscheinen u. a. RND(x) mit drei verschiedenen Zufallszahlengeneratoren, CLK(x) zur Tageszeitangabe in

32 IBM. IBM System/360 OS/DOS ITF: BASIC Terminal User's Guide. 2nd ed., New York 1970; IBM. IBM System/360 OS (TSO) ITF: BASIC Terminal User's Guide. New York 1971; IBM VS BASIC Language. 3rd ed., New York 1976.

33 IBM. CALL: BASIC Language Reference Manual. 3rd ed., New York 1973; IBM. CALL-OS: BASIC Language Reference Manual. 4th ed., New York 1972; IBM. CALL DCS: BASIC Language Introduction. 2nd ed., New York 1972; IBM. CALL: BASIC Einführung. Stuttgart 1970; IBM. CALL: BASIC Handbuch. Stuttgart 1971.

34 IBM. Business Analysis/BASIC. General Information Manual. 3rd., New York 1973.

35 IBM. BASIC-Handbuch für den Informatik-Unterricht in Schulen. Bonn-Bad Godesberg 1973.

36 Philips. P/9200 Time Sharing BASIC. Apeldoorn 1973. Philips. P9200 Time Sharing System. Reference Card BASIC. Apeldoorn 1973; Philips. P/9200 Time Sharing System. BASIC. Düsseldorf 1971; Philips. P800M BASIC Manual. Apeldoorn 1974.

Stunden und Minuten, TIM(x) zur Erfassung der Programmablaufzeit und TAB(x) zur Vornahme von Tabulatorsprüngen.
Der P9200 BASIC-Compiler übersetzt im Mittel 100 Anweisungen pro Sekunde; die maximale Länge eines BASIC-Programms beträgt 700 Statements (Programmzeilen) bei einer durchschnittlichen Länge von 40 Zeichen pro Statement.
Benutzer des Philips Time-Sharing-Systems haben Zugriff auf eine reichhaltige Programmbibliothek, die auch Statistik-, OR- und kommerzielle Programme umfaßt.[37]

112.5 Computer Gesellschaft Konstanz mbH

Das System TR 440 lässt als Dialogsprache auch BASIC zu, wobei anfänglich auf möglichst weitgehende Übereinstimmung mit Dartmouth BASIC Wert gelegt wurde. Dies äussert sich darin, dass die – noch von der Telefunken Computer GmbH – Benutzern der TR 440 zur Verfügung gestellten BASIC-Sprachbeschreibungen in ihrer Ursprungsversion mit entsprechenden Publikationen von GE und Honeywell Bull praktisch identisch waren. Abweichungen bestanden nur bezüglich der Systembefehle; ausserdem erfolgten die Fehlermeldungen in deutscher Sprache. Eine zunehmende Verselbständigung von TR BASIC lässt sich anhand der 1974 und 1975 erschienenen Nachträge zum BASIC Manual feststellen.[38]

112.6 National Cash Register Company

Der NCR Century Computer verfügt über einen leistungsfähigen BASIC-Compiler, der den für ANSI Minimal BASIC festgelegten Anforderungen weitgehend entspricht. Insbesondere verfügt der NCR BASIC-Compiler über alle wesentlichen Funktionen (RND ohne Argumentangabe), das RANDOMIZE-Statement, usw. Darüber hinaus lässt der NCR BASIC-Compiler die Verwendung alternativer Vergleichsoperatoren (= < oder < =, usw.) zu und bietet die Möglichkeit zum Rechnen mit einfacher und doppelter Genauigkeit. Der Compiler beinhaltet auch

37 Philips. P/9200 Time Sharing System. Programm-Bibliothek. Übersicht. Düsseldorf 1972.
38 Computer Gesellschaft Konstanz. System TR 440. BASIC Sprachbeschreibung. Konstanz 1974. – Ursprungsversion (1972) von Telefunken Computer GmbH.

computerinterne numerische Konstante (Pi = 3.1416 und e = 2.7183), PAGE-Statement, eine Variante zum REM-Statement, alternativ benutzbares ON/GOTO resp. ON/THEN-Statement. Besonders leistungsfähig sind – wie bei Honeywell Bull – die zur Dateiverarbeitung verfügbaren Statements (RESTORE, RESTORE*, RESTORE$ usw.).[39]

112.7 International Computer Limited

ICL Computer der Serien 1900 und 2900 verfügen ebenfalls über einen BASIC-Compiler. ICL 1900 BASIC entspricht im wesentlichen Dartmouth BASIC-1; leistungsfähiger ist der ICL 2900 BASIC-Compiler, der nahezu vollumfänglich den Standards für ANSI Minimal BASIC entspricht und teilweise beträchtlich übersteigt.[40]

BASIC ist eine in der Grundform weitgehend standardisierte Programmiersprache.

112.8 BASIC Timesharing, Inc.

BASIC wurde in den USA auch durch Timesharing-Dienstleistungsunternehmungen weiterentwickelt. Erwähnt sei die BASIC Timesharing, Inc., deren BASIC-X (extended) auch die Möglichkeit zur Matrizen- und Fileverarbeitung bietet, wobei insbesondere die dateibezogenen Anweisungen recht vielfältig sind.[41] Weiterhin zeichnet sich BASIC-X durch folgende Besonderheiten aus:

- PRINT USING-Statement mit freier Wahl von I-, F- oder E-Format für numerische Variable und Masken für alphanumerische Variable.

39 National Cash Register Company. NCR Century BASIC-2. Student Text. Dayton 1973.

40 ICL. 1900 Series. BASIC Reference Manual. London 1971.

41 Verfügbar sind die BTI Systeme 3000 und 4000. "The BTI 3000 is a... versatile timesharing system capable of operating 32 ports in a considerably extended version of BASIC. Language features include matrix and string manipulation, string arithmetic with 252-digit precision, and sophisticated file handling. The BTI 3000 offers a powerful management structure and massive file capacity, expandable to 4 billion bytes." BASIC Timesharing, Inc. Model 3000 Interactive Timesharing System. Sunnyvale 1974, p. 2.

- Möglichkeit zur sukzessiven Programmbearbeitung im Execute Immediate-Modus, mit sofortiger statementbezogener Fehlererkennung.[42]

112.9 Com-Share Incorporated

In noch stärkerem Masse wurde BASIC durch Com-Share Inc. weiterentwickelt und als NEWBASIC auf dem Computersystem XDS 940 implementiert.[43]

NEWBASIC weist folgende Besonderheiten auf:[44]

- Spezialkonstante, PI = 3.141592654;
- fünf Datentypen: INTEGER, DOUBLE INTEGER, REAL, DOUBLE REAL, COMPLEX;
- zusätzliche Vergleichsoperatoren, beispielsweise auch für <<, >> und = ≠ (ungefähr gleich);
- zusätzliche Boolsche Operatoren: XOR (logical exclusive OR), IMP (logical implication), EQU (logical equivalence), BUT (logical conjunction), BAN (binary conjunction), BOR (binary disjunction), BEX (binary exclusive OR);
- Alternativform für REM-Statement, mit ! und unmittelbarer Anschlußmöglichkeit an andere Statements;
- implizites und/oder mehrfaches LET-Statement;
- globale Nullsetzung aller Programmvariablen, mit VAR = ZERO;
- flexible Schleifenbearbeitung, mit FOR/einfaches oder multiples NEXT, FOR/UNTIL, FOR/WHILE;
- vollständig bedingte Anweisungen, mit IF/THEN/ELSE;
- zusätzliche Programmsteuerungsanweisungen, insbesondere ON ENDFILE, ON ERROR, ON ESCAPE;
- mehrzeilige Benutzerfunktionen, definiert innerhalb DEF und RETURN;
- CALL-Statement für mehrzeilige Benutzerfunktionen;
- flexible Druckgestaltung;
- Spezialbefehle zur Programmbereinigung (debugging), insbesondere AGAIN, STEP, TRACE.

NEWBASIC wird insbesondere in Verbindung mit dem Projekt Solo der

42 BASIC Timesharing, Inc. Model 4000 Interactive Timesharing System – User's Manual. Sunnyvale 1975.
43 XDS = Xerox Data Systems; vgl. Abs. 122.5.
44 Com-Share Incorporated. NEWBASIC Reference Manual. Pittsburgh 1971.

Universität Pittsburgh zum computerunterstützten Unterricht verwendet.[45]
Die für NEWBASIC charakteristische starke Ausweitung der ursprünglichen BASIC-Version des Dartmouth College wurde in der Fachliteratur teilweise sehr zurückhaltend beurteilt. Zusammenfassend stellte Jerry L. Odgin fest: »The BASIC language (Beginner's All-purpose Symbolic Instruction Code) ... was developed for undergraduates at the college algebra level, and was obviously designed on the premise that something had to be easier to teach than FORTRAN. ... Since many system programmers have had experience with FORTRAN, they have attempted to make BASIC look more and more like FORTRAN. It is unreasonable. Extensions to the BASIC language which maintain its simplicity yet widen the scope of applicability are possible. I feel that some of the extensions have been absurd.«[46] Einzelne Sprachausweitungen führten offensichtlich zu einer Beeinträchtigung des lernorientierten Charakters von BASIC.

12 BASIC Monologbetrieb

BASIC wurde an amerikanischen Hochschulen schon frühzeitig bei reiner Stapelverarbeitung (batch processing) als lernorientierte Programmiersprache verwendet. Dabei gelangte zumeist die für den Monologbetrieb besonders geeignete BASIC-Version der University of Washington zum Einsatz.

BASIC ist auch als Monologsprache verwendbar.

121 Version der University of Washington

Der erste ausschliesslich für Monologbetrieb konzipierte BASIC-Compiler wurde an der Graduate School of Business Administration der University of Washington unter der Leitung von William F. Sharpe entwickelt

45 Thomas A. Dwyer. Some Principles for the Human Use of Computers in Education. International Journal of Man-Machine Studies 3 (1971) 3, July, p. 219–239; Thomas A. Dwyer. Teacher/Student Authored CAI Using the NEWBASIC System. Communications of the ACM 15 (1972) 1, January, p. 21–28. – Vgl. auch: Project Solo, University of Pittsburgh. Solo Works: Newsletter.

46 Jerry L. Odgin. The Case Against BASIC. Datamation 17 (1971) 17, September 1, p. 34–41.

und 1967 auf der dortigen Datenverarbeitungsanlage (IBM 7094) implementiert.
Die besondere Eigenart des University of Washington BASIC Interpretive Compiler (UWBIC) bestand darin, dass er vollständig in FORTRAN IV programmiert war[47] und als Compiler-Compiler leicht auf grössere Datenverarbeitungsanlagen beliebigen Typs übernommen werden konnte.[48]

UWBIC ist ein BASIC-Compiler auf FORTRAN-Basis.

Im Vergleich zum Dartmouth BASIC-Prozessor war die UWBIC-Leistungsfähigkeit dadurch beschränkt, dass
- keine Matrizenoperationen,
- keine logischen Operationen,
- keine Zeichenkettenverarbeitung und
- keine Dateiverarbeitung

möglich waren.[49]
Diese Beschränkungen erwiesen sich insofern nicht als gravierend, als UW BASIC primär Studenten der Wirtschaftswissenschaften mit den Grundkonzepten der Programmierung vertraut machen und ihnen den späteren Einstieg in andere Programmiersprachen, insbesondere FORTRAN, erleichtern sollte.
Im Hinblick auf diese Grundziele wurde auch auf besonders leicht verständliche Fehlermeldungen (diagnostic messages) Wert gelegt. Die folgende Liste zeigt einige besonders charakteristische Fehlerhinweise.

```
PROGRAM AND DATA EXCEED AVAILABLE STORAGE AT THIS POINT
SORRY - I ACCEPT ONLY LINE NUMBERS BETWEEN 1 AND  9999
I CANNOT FIND AN EQUAL SIGN IN THE LET STATEMENT ABOVE
THE EQUAL SIGN ABOVE IS IN A FUNNY PLACE
ITEM TO THE LEFT OF THE EQUAL SIGN MUST BE A VARIABLE
THE QUOTATION MARKS ABOVE ARE NOT PAIRED
THE IF STATEMENT ABOVE IS HOPELESS
```

47 William F. Sharpe. University of Washington BASIC Interpretive Compiler UWBIC. Seattle 1967.
48 "Although used on an I.B.M. 7040–7094 computer system, the University of Washington BASIC Interpretive Compiler is sufficiently general to be used on any medium of any large scale computer with FORTRAN IV compilers." Couger. Computers and the Schools of Business, p. 74.
49 Vgl. William F. Sharpe and Nancy L. Jacob. BASIC. An Introduction to Computer Programming Using the BASIC Language. Rev. ed., New York 1971; Richard L. Nolan. Introduction to Computing Through the BASIC Language. New York 1969.

```
I CANNOT ACCEPT AN IF STATEMENT WITHOUT A COMPARISON
I CANNOT FIND THEN OR GOTO
THE FOR STATEMENT ABOVE IS HOPELESS
I CANNOT FIND AN EQUAL SIGN IN THE FOR STATEMENT ABOVE
THE VARIABLE IN A FOR STATEMENT MUST BE UNSUBSCRIPTED
DIM STATEMENT INCORRECTLY WRITTEN
YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
STATEMENT NUMBER    0 APPEARS MORE THAN ONCE
I EXPECTED YOU TO MENTION VARIABLE 00 -- I WILL ASSUME THAT YOU DID
THERE ARE NO DATA IN THIS DATA STATEMENT
PROGRAM STOPPED -- I CANNOT GO TO   12 BECAUSE IT IS NOT THERE
PROGRAM STOPPED -- I CANNOT RETURN SINCE I DID NOT COME FROM A GOSUB
```

UWBIC wies im Vergleich zu FORTRAN eine Reihe von – insbesondere für Anfänger wichtigen – Vorteilen auf.[50] UWBIC fand deshalb an amerikanischen Hochschulen relativ rasche Verbreitung. So wurde UWBIC bereits 1968 am College of Commerce and Business Administration der University of Illinois in Urbana-Champaign eingesetzt.

122 Weiterentwicklungen

122.1 Worcester Polytechnic Institute

Mit dem UWBIC-Compiler wurde erstmals ein modular aufgebauter Compiler-Compiler als Übersetzer für eine vollentwickelte Programmiersprache geschaffen. UWBIC war deshalb auch als Instruktionsmaterial zum Compilerbau besonders geeignet. Entsprechendes gilt auch für den am Worcester Polytechnik Institute entwickelten VIEWIT-Compiler.[51]
VIEWIT kann als BASIC-Untersuchungscompiler charakterisiert werden. Er vermag insgesamt vierzehn BASIC-Statements[52] stufenweise –

50 Vgl. Karl Weber und Carl Wolfram Türschmann. BASIC 1. Lehr- und Handbuch der Programmiersprache BASIC mit wirtschaftswissenschaftlichen Anwendungsbeispielen. 1. Aufl., Bern 1977, S. 33.
51 Elaine M. Henshon. VIEWIT: A Take-Apart Compiler. M. S. Thesis, Worcester Polytechnic Institute. Worcester 1973.
52 REM, DIM, DEF, DATA, READ, PRINT, LET, GOTO, IF, FOR, NEXT, END, GOSUB, RETURN.

als "lexical analyzer" und "syntax/semantics analyzer" – zu analysieren.[53] Eine Programmausführung ist mit dem VIEWIT-Compiler nicht möglich.

122.2 Justus-Liebig-Universität Giessen

Der University of Washington BASIC Interpretive Compiler (UWBIC) wurde 1970 von der Professur für Betriebswirtschaftslehre V auf die Rechenanlage der Justus Liebig-Universität Giessen (CD 3300) übernommen.
Ausgehend von den von Professor Weber an der Universität Illinois gesammelten praktischen Erfahrungen, wurde UW BASIC zunächst in der Lehre verwendet und der UWBIC-Compiler in der aus Abbildung 2 ersichtlichen Weise eingesetzt.[54] Besonderes Gewicht wurde hierbei auf eine gruppenweise Verarbeitung von BASIC-Programmen gelegt, wobei die einzelnen Benutzer lediglich ihre Programm-(und Daten-)Karten mit den entsprechenden BASIC-Start- und -Endekarten an eine zentrale Sammelstelle einreichen mussten und die bearbeiteten Jobs auch dort mit den Ausgabelisten (Fehlermeldungen/Resultate) wieder in Empfang nehmen konnten.
In den Jahren 1970–1974 wurde UWBIC an der Professur für Betriebswirtschaftslehre V weiterentwickelt und dadurch FOSBIC (**FO**RTRAN **S**imulated **B**ASIC **I**nterpretive **C**ompiler) geschaffen.

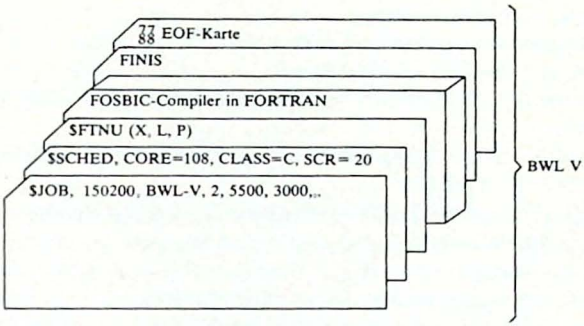
FOSBIC ist ein modular aufgebauter BASIC Compiler-Compiler.

Weiterhin kann FOSBIC als ein sich durch hohe Hardware- und Software-Kompatibilität auszeichnender, benutzerfreundlicher und transportfähiger Compiler charakterisiert werden.

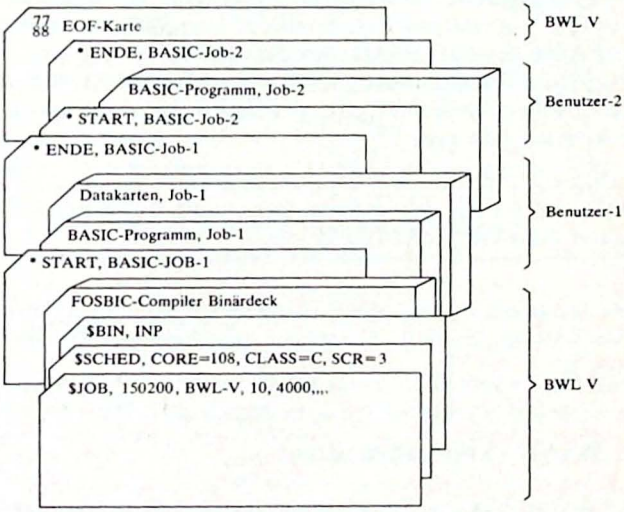
53 "The VIEWIT compiler is designed and documented to serve as a model for the compiling process. It supports a modified version of the BASIC source language which offers reasonable processing power while being simular enough to keep the compiler design straight-forward. ... The VIEWIT compiler is implanted in FORTRAN. In addition, the various functional routines of VIEWIT are highly modular in construction so that they can be easily replaced to allow more sophisticated alorithms to be used in the compilation or to allow extensions to the original source language." VIEWIT: A Take-Apart Compiler. Computing Newsletter for Schools of Business 8 (1974) 4, December, p. 7.

54 Anstelle der Bezeichnung UWBIC ist in Abbildung 2 bereits die Bezeichnung FOSBIC (FORTRAN Simulated BASIC Interpretive Compiler) eingesetzt.

Abbildung 2. Verarbeitung von BASIC-Programmen, Grundversion BWL V für CD 3300



Phase 1: Erstellung des FOSBIC-Compiler-Binärdecks aufgrund des FOSBIC-Compiler-FORTRAN-Programms



Phase 2. Abarbeitung von BASIC-Programmen unter Verwendung des FOSBIC-Compiler-Binärdecks

FOSBIC weist – im Vergleich zu UWBIC folgende benutzerrelevante Besonderheiten auf:

- Steuerungsmöglichkeit der Druckbreite und Felderzahl durch den Benutzer;
- nicht compiliertes COMMENT-Statement;
- Zeichenkettenverarbeitung;
- Eingabe grosser Zahlen in E-Format;
- Texteingabe via DATA-Statement;
- Verarbeitung unterschiedlicher Statementformen gleicher Logik
  LET, let
  GOTO/ON, GOTO/OF, ON/GOTO, OF/GOTO
  IF/GOTO, IF/THEN
  GOSUB/ON, GOSUB/OF, ON/GOSUB, OF/GOSUB;
- Matrizenverarbeitung;
- Fileverarbeitung;
- Dateiverarbeitungssimulation, mit RESTORE.

Damit kann FOSBIC im wesentlichen alle Operationen des Dartmouth BASIC-Compilers übernehmen.
Der FOSBIC-Compiler wurde an der Justus Liebig-Universität Giessen 1974 in binärer Form auf eine Magnetplatte installiert und als Standard-Software freigegeben. Damit hatte der einzelne Benutzer die Möglichkeit, mit fünf Systemsteuerkarten auf den Compiler zuzugreifen und beliebige BASIC-Jobs frei bearbeiten zu lassen.
Der FOSBIC-Compiler wurde 1977 vollumfänglich veröffentlicht[55] Er gilt nach Professor William F. Sharpe (Stanford University) als "a system of the most modern type".[56]

FOSBIC ist ein zur allgemeinen Benutzung freigegebener BASIC-Compiler auf FORTRAN-Basis.

1980 wurde an der Justus-Liebig-Universität Giessen eine weiterentwickelte Version des FOSBIC-Compilers implementiert (Cyber 174). Vgl. Abs. 132.8.

13 BASIC-Alternativbetrieb

Andere BASIC-Entwicklungen gehen dahin, die – ursprünglich für den Dialogverkehr konzipierte – Programmsprache auch alternativ zur Da-

55 Karl Weber und Carl Wolfram Türschmann. FOSBIC-Compiler. Ein BASIC-Compilersystem auf FORTRAN-Basis. Bern und Stuttgart 1977.
56 Brief vom 30. Januar 1974.

tenverarbeitung im Batch-Processing-Mode verwendbar zu machen. Diese Entwicklung wurde an der University of Maryland eingeleitet und vermochte sich in der Folge weitgehend durchzusetzen.

BASIC kann vielfach auch alternativ im Dialog- und Stapelbetrieb verwendet werden.

131 Version der University of Maryland

An der Universität of Maryland wurde 1968 mit Unterstützung der National Aeronautics and Space Administration der UNIVAC 1108 BASIC Processor (UOM BASIC) entwickelt. Auf Dartmouth BASIC basierend war er primär für Dialogbetrieb (conversational use) konzipiert, ließ aber auch den Stapelbetrieb (batch processing) zu.[57]
UOM BASIC wies im übrigen in der ersten Version folgende Besonderheiten auf:

- Alternativ verwendbare Vergleichsoperatoren;
- Umwandlungsmöglichkeit von numerischen/alphanumerischen Variablen, mit CHANGE;
- variable Startbasis für Zufallszahlengenerator, durch RANDOMIZE;
- Alternativform für REM-Statement, mit unmittelbarer Anschlußmöglichkeit an andere Anweisungen;
- implizites let-Statement;
- mehrzeilige Benutzerfunktionen, definiert innerhalb DEF und FNEND;
- CALL-Statement für mehrzeilige Benutzerfunktionen;
- Alternativform für FOR/NEXT-Statement.

132 Weiterentwicklungen

Im Alternativbetrieb werden im allgemeinen für Dialog- und Monologbetrieb unterschiedliche Sprachprozessoren (Interpreter/Compiler) eingesetzt.

132.1 Sperry Rand Corporation

Der UOM BASIC-Compiler wurde in der Folge von der Sperry Rand Corporation für das Computersystem 1100 übernommen. Die nunmehr

57 George Ohlmacher, Robert Cassels, Thomas Hall, Brian Stanley und Alfred Beams. BASIC for the UNIVAC 1108. College Park 1968.

einem weiteren Benutzerkreis verfügbare BASIC-Version wurde neu bezeichnet, die ursprünglichen UOM BASIC-Manuale aber praktisch unverändert rezipiert.[58]

132.2 Burroughs Corporation

Von der Burroughs Corporation wurde BASIC für die Systeme B 5500 und B 6700 implementiert. Die verwendete BASIC-Version weicht von der BASIC-Grundform des Dartmouth College nur durch Zulassung einiger Spezialanweisungen ab. Besonders zu vermerken ist die Möglichkeit zur ganzzahligen Division (dyadischer Operator DIV) und Restwertausweisung (dyadischer Operator MOD), wobei generell
A MOD B = A − (A DIV B)*B gilt.
Beispiel: 9 DIV 4 = 2; 9 MOD 4 = 1.
Bezüglich des Batch-Betriebs ist beachtenswert, dass die Dateneingabe über Kartenleser oder Magnetbandeinheit erfolgen kann. Die Datenausgabe geschieht normalerweise (mit vorheriger Zwischenspeicherung auf Magnetplatte) über einen Schnelldrucker.[59]

132.3 CII Honeywell Bull

Die CII Honeywell Bull entstand Ende 1975 durch Fusion der französischen Compagnie Internationale pour l'Informatique (CII) mit der Honeywell Bull Company.[60]
Bislang wurde BASIC von der CII insbesondere im Rahmen der SIRIS 7/SIRIS 8 Time Sharing Systeme verwendet.[61] Dabei konnte auf BASIC-Erfahrungen mit dem Datenverarbeitungssystem 10070 zurückgegriffen werden, wobei für dieses System die BASIC-Version von Xerox Data System (BTM BASIC) und die zugehörigen BASIC-Handbücher in unveränderter Form benutzt wurden.[62]

58 Sperry Rand. Sperry UNIVAC 1100 Series. BASIC. Rev. 2, St. Paul 1978.
59 Burroughs. B5500 Time Sharing System. BASIC Reference Manual. Detroit 1968; Burroughs. B6700 BASIC Language. Information Manual. Detroit 1971.
60 Französisch-amerikanische Kapitalbeteiligung: 53% resp. 47%. – Vgl. auch Abs. 122.2.
61 Compagnie Internationale pour l'Informatique. 10070 BASIC User's Manual. Louveciennes 1969.
62 Compagnie Internationale pour l'Informatique. BASIC under SIRIS 7/SIRIS 8 Time Sharing. Operation and User's Manual. Louveciennes 1973.

Besonders beachtenswert ist das – auch in XDS BASIC[63] verfügbare – SIM-Statement, mit dessen Hilfe lineare Gleichungssysteme der Form AX = B (nach vorheriger Einlesung der Matrizen A und B, beispielsweise mittels MAT READ A, B) mit der Anweisung MAT X = SIM (A, B) gelöst (und die Resultate mit MAT PRINT X ausgegeben) werden können.[64]

132.4 Control Data Corporation

BASIC ist für die Computersysteme der Serie CD Cyber allgemein verfügbar. Besonders leistungsfähig sind die Versionen KRONOS III und BASIC 3.[65]

132.5 Siemens Aktiengesellschaft

Das Siemens-Teilnehmersystem BSV 4004 lässt als Programmiersprache ebenfalls BASIC zu. Neben der üblichen Methode, BASIC-Programme im Dialog zu erstellen und zu testen, besteht auch die Möglichkeit zum off-line Betrieb unter Verwendung von Lochkarten oder Lochstreifen.[66] Benutzern des Siemens-Teilnehmersystems BSV 4004 steht auch das Programmpaket DIBA zur Verfügung; es umfasst eine grosse Zahl von **DI**alog-**B**ASIC-**A**nwenderprogrammen, insbesondere auch aus den Gebieten von Mathematik, Statistik und Betriebswirtschaftslehre.[67] Als Besonderheit sei vermerkt, dass sämtliche DIBA in der Maschinensprache vorliegen und damit die Möglichkeit zur Vornahme benutzereigener Programmodifikationen entfällt.
Dialog-BASIC ist auch für das Siemens-Betriebssystem BS 2000 verfügbar.[68]

63 Xerox Data Systems. Xerox BASIC Sigma 5–9 Computers, p. 22.
64 Compagnie Internationale pour l'Informatique. 10070 BASIC User's Manual, p. 20; Compagnie Internationale pour l'Informatique. BASIC under SIRIS 7/SIRIS 8 Time Sharing, p. 4.2–3.
65 Control Data. KRONOS III. BASIC Reference Manual. St. Paul 1970; Control Data. BASIC Version 3. Reference Manual. Sunnyvale 1979. – Vgl. auch Weber und Türschmann. BASIC 1, 1. Aufl., S. 26.
66 Andreas Alteneder und Christel Offelder. BASIC-Praktikum. Lernprogramm. München 1971; Andreas Alteneder und Christel Offelder. BASIC-Praktikum. Katalog. München 1971.
67 Siemens. Siemens System 4004. DIBA Dialog-BASIC-Anwenderprogramme. Teil I–III. München 1971/72.
68 Siemens. Betriebssystem BS 2000. Dialog-BASIC. Beschreibung. 3. Aufl., München 1979.

132.6 Xerox Data System

Die XDS Rechenanlagen Sigma 5–9 verfügen über einen BASIC-Compiler, der in besonderem Masse für Alternativbetrieb geeignet ist. Bei allen Anlagen kann mit dem Batch Processing Monitor (BTM oder BPM) und bei den Systemen Sigma 6/7/9 mit dem leistungsfähigeren Universal Time-Sharing Monitor (UTS) gearbeitet werden.[69]
UTS BASIC weist – im Vergleich zu den früheren Versionen von Dartmouth BASIC – folgende Besonderheiten auf:

- Möglichkeit zur Variation der Druckbreite (72 Druckstellen normal), mit WIDTH;
- Wahl zwischen einfacher und doppelter Rechengenauigkeit, ohne/mit ENTER BASIC L;
- generelle oder selektive Druckausgabe der BASIC-Statements, mit LIST resp. LIST mit Zeilenvermerk;
- unterschiedliche Prüfungsintensität bei der Kompilation, RUN mit Dimensionsprüfung für alle Variable (safe mode) und FAST ohne Prüfung;
- Einlesung programmexterner Daten, mit INPUT (anstelle von READ/DATA).

Die vorerwähnten Modalitäten können durch entsprechende Steuerkarten festgelegt werden.
Weiterhin weist UTS BASIC folgende Merkmale auf:

- Zusätzliche Standardfunktionen, insbesondere DEG und RAD zur Umrechnung von Gradmass in Bogenmass und umgekehrt;
- spezielle Wertzuweisungsanweisungen, insbesondere CLEAR, CLEAR ARRAYS, CLEAR STRINGS zur Speicherplatzlöschung und NULL, NULL ARRAYS, NULL SIMVARS (simple variables) zur generellen resp. gezielten Nullsetzung von Speicherplätzen;
- unterschiedliche Formen der Programmspeicherung, mit FILE (execution mode) und FILE PACK (editing mode).

132.7 International Timesharing Corporation

Von der International Timesharing Corporation wurde – in Zusammenarbeit mit der Control Data Corporation – die ursprüngliche BASIC-Version wohl am stärksten an die Programmiersprachen FORTRAN und

69 Xerox Data Systems. Xerox BASIC Sigma 5–9 Computers. Language and Operations Reference Manual. El Segundo 1971.

COBOL angenähert. Die damit entstandene "extended version of BASIC" wird als EXBASIC bezeichnet.[70]

I T S BASIC-Programme können direkt über einen Terminal oder mit Hilfe von Lochstreifen eingegeben werden, wobei die Programmübersetzung entweder schrittweise (conversational mode) oder gesamthaft (production mode) erfolgen kann. Bei der letzterwähnten – zeitsparenden – Methode kann das Objektprogramm in Binärform plattengespeichert werden, so dass bei nochmaliger Verwendung die Rekompilation entfällt. Die besondere Eigenart von EXBASIC ergibt sich aus folgenden Möglichkeiten:

- Verwendung mehrzeiliger BASIC-Statements, maximale Zeichenzahl 251;
- Veränderung der Druckstellenzahl im Programmablauf, mit WIDTH;
- freiformatierte Dateneingabe und -Ausgabe, mit FORMAT;
- Druckaufbereitungseintragungen, mit FORMAT;
- maskengebundene Datenausgabe, mit IMAGE.

Die Formatspezifikationen können bei I T S EXBASIC wie folgt vorgenommen werden, falls w die Anzahl der Stellen insgesamt, d die Anzahl der Stellen hinter dem Dezimalpunkt und r den Wiederholungsfaktor darstellen:

rEw.d. – E-Format für Real-Zahlen in Exponentiationsschreibweise
rFw.d – F-Format für Real-Zahlen in Gleitkommadarstellung
rIw – I-Format für Integer-Zahlen
rOw – O-Format für Oktalzahlen
rAw – A-Format für Textverarbeitung
rCw – C-Format für Kontrollzeichen

Weiterhin sind zulässig:
nHs – H-Format für Textausgabe
rX – X-Format für Leerstellen
„S" – S-Format für Strings
/ – für Zeilenschaltung

Die vorerwähnten Formatangaben stimmen vollständig mit den Formatangaben von ANSI FORTRAN IV überein, wobei lediglich das D- und G-Format sowie der Skalierungsfaktor P fehlen.

70 International Timesharing Corporation. I T S EXBASIC. Reference Manual. Chaska 1971. – Hinweis von Herr David C. Kistle, Information Services, I T S International Timesharing Corporation, Chaska, Minnesota (vom 13. Juni 1974): "I T S EXBASIC was developed specificially for the I T S timesharing system. The I T S timesharing system uses two Control Data Corporation 3300 computer systems. The services of our timesharing system are available in 20 cities throughout the United States."

Mit einem IMAGE-Statement können folgende Eintragungen verbunden sein:

####	– für Zahlen in E-Format
%%.%%	– für Zahlen in F-Format
%%	– für Zahlen in I-Format
$$.$$	– für Währungsvorzeichen
****	– für Schutzsternschreibung
%%à%	– Prozentzeichen

ITS EXBASIC entspricht der für CD 6000 Systeme verfügbaren Version KRONOS III der Control Data Corporation.[71] Diese Weiterentwicklungen führen – durch die Sprachannäherung an FORTRAN und COBOL – zu einer Überdeckung des ursprünglichen lernorientierten Charakters von BASIC, was in der amerikanischen Literatur nicht nur positiv beurteilt wird.[72]

132.8 Justus-Liebig-Universität Giessen

Die Ursprungsversion des FOSBIC-Compilers wurde seit 1977 stetig weiterentwickelt. Die 1980 auf der CD Cyber 174 der Justus-Liebig-Universität Giessen implementierte FOSBIC-Version 2 besitzt gegenüber der Ursprungsversion[73] eine Vielzahl von Erweiterungen. Verwiesen sei insbesondere auf

- zusätzliche Anweisungen (WIDTH-, DIGITS-, multiples LET-Statement, DEF-Statement),
- zusätzlich eingebaute Funktionen.

Ausserdem gewährleistet das umfangreichere BASIC-Programmiersystem eine

- verbesserte Dialogfähigkeit und
- übersichtlichere Fileverarbeitung.

Besonders zu erwähnen ist, dass die FOSBIC-Version 2 kein speziell auf die Kennworte der Programmiersprache BASIC abgestimmter Compiler

71 Control Data Corporation. KRONOS III. BASIC Reference Manual. St. Paul 1970.

72 "In the blind objective of making BASIC suit all uses and all users, unnecessary and unreasonable changes have been perpetrated on the flimsly premise that all potential programs should be able to be programmed in BASIC. ... That the fate of divergent development should befall such an elegantly simple language is a crime." Jerry L. Odgin. The Case Against BASIC. Datamation 17 (1971) 17, September 1, p. 41 und p. 34.

73 Vgl. Karl Weber und Carl Wolfram Türschmann. FOSBIC-Compiler. Ein BASIC-Compilersystem auf FORTRAN-Basis. Bern 1977.

darstellt, sondern jede beliebige BASIC-ähnliche Sprachstruktur verarbeiten kann. Dadurch besteht zum Beispiel die Möglichkeit, die Kennwortstruktur von BASIC sowie alle möglichen Fehlermeldungen in eine beliebige Sprache zu übersetzen und damit die Bindung an die englische Sprache aufzuheben.[74] Statt der BASIC-Anweisung

```
210 PRINT "DIES IST EIN VERSUCH"
```

kann bei entsprechender Implementierung der FOSBIC-Version 2 auch die analoge Anweisung

```
210 DRUCKE "DIES IST EIN VERSUCH"
```

verstanden und richtig ausgeführt werden.
Diese Erweiterung gestattet es, den FOSBIC-Compiler in Grundschulklassen einzusetzen, deren Englischkenntnisse nicht ausreichen, um die ursprünglichen Kennworte der Programmiersprache BASIC und eventuell die dazugehörigen Fehlermeldungen zu verstehen.
Das nachfolgende Programmbeispiel verdeutlicht diese hochflexible sprachstrukturelle Möglichkeit des FOSBIC-Compilers mit einer Reihe von Elementaranweisungen.

```
 10  BEMERKUNG GLEICHUNG ZWEITEN GRADES
 20  DRUCKE "          A","          B","          C","          X(1)","          X(2)",
 25  DRUCKE " BEMERKUNG"
 30  LESE A,B,C
 40  WENN A <> 0 GEH NACH 80
 50  WENN B <> 0 GEH NACH 140
 60  WENN C <> 0 GEH NACH 190
 70  GEH NACH 900
 80  RECHNE D = B**2-4*A*C
 90  WENN D < 0 GEH NACH 170
100  RECHNE X1 = (-B+SQR(D))/(2*A)
110  RECHNE X2 = (-B-SQR(D))/(2*A)
120  DRUCKE A,B,C,X1,X2," REELL"
130  GEH NACH 30
140  RECHNE X = -C/B
150  DRUCKE A,B,C,X," "," LINEAR"
160  GEH NACH 30
170  DRUCKE A,B,C," "," "," KOMPLEX"
180  GEH NACH 30
190  DRUCKE "FALSCHE DATENEINGABE"
200  GEH NACH 30
210  DATEN -1, 6, 2, 3, 1, 1, 1, 0, 0, 0
220  BEMERKUNG SIEHE ABBILDUNG 11 UND KAPITEL 111
900  HALT
950  END
```

Das Beispiel ist identisch mit dem BASIC-Programm auf S. 17.
Da eine sprachstrukturelle Veränderung keine Neuinstallation des FOSBIC-Compilersystems verlangt, kann innerhalb eines Kurses die Sprach-

74 Zu ähnlichen Entwicklungstendenzen vgl. auch: Pierre Lebeaux. Introduction au BASIC. Paris 1980, p. 63–127.

struktur kontinuierlich den fortschreitenden Ansprüchen der Benutzer angepasst werden.
Die Entwicklung des FOSBIC-Compilersystems ist damit noch nicht abgeschlossen. Zukünftige Forschungsarbeiten zielen darauf hin

- betriebswirtschaftliche Methoden und Modelle in die Programmiersprache BASIC als Module einzubauen,
- ein datenunabhängiges Datenbanksystem einzufügen und
- lehrorientierte Module zu integrieren, die es dem Benutzer gestatten, direkt vom Programmiersystem selbst in die Anwendung eingeführt zu werden.

Es ist geplant, auch diese Forschungsarbeiten nach ihrem Abschluss zu veröffentlichen und so das weiterentwickelte FOSBIC-Compilersystem allgemein zugänglich zu machen.
Die Installation des FOSBIC-Compilersystems auf der Cyber 174 erfolgt durch die nachstehende Anweisungsfolge in CCL-Sprache des Maschinensystems (CCL = **C**omputer **C**ontrol **L**anguage).

FOSBIC-Compiler Installation, CD Cyber 174

```
FOSBIC,T200.
USER,150210,XXXX.
CHARGE,150210,150210.
BANNER.   FOSBIC-VERSION 2.1
ATTACH,NEWBAS.
ATTACH,FOSBIC/M=W.
FTN,I=NEWBAS.
LOAD(LGO).
NOGO(FOSBIC).
<EOI>
```

Die Installation kann auch im Dialogbetrieb mit nahezu identischen Anweisungen durchgeführt werden. Lediglich der Systembeginn gestaltet sich etwas anders, was aus dem Aufruf des FOSBIC-Compilersystems im Dialog zu ersehen ist. Bei dieser Installationsform steht das Quellendeck der FORTRAN-Anweisungen des FOSBIC-Compilersystems auf dem Plattenfile mit dem Namen NEWBAS. Das übersetzte und maschinenorientierte Binärdeck wird auf das Plattenfile mit dem Namen FOSBIC geschrieben, wobei die Kennung FOSBIC immer notwendig ist, da die OVERLAY-Teile des FOSBIC-Compilersystems so bezeichnet sind.
Nach dieser Installationsphase kann der FOSBIC-Compiler sowohl im Monolog- als auch im Dialogbetrieb aufgerufen werden.

FOSBIC-Monologaufruf, CD Cyber 174

```
MONOLOG,T1000.
USER,150210,XXXX.
CHARGE,150210,150210.
BANNER.   FOSBIC-MONOLOGAUFRUF
```

```
ATTACH,FOSBIC.
ATTACH,TAPE4=FEHLER.
ATTACH,TAPE5=INFO.
ATTACH,TAPE6=BENBIBL/M=W
FOSBIC.
<EOR>-KARTE
* JOB-KARTE DES ERSTEN BASIC-JOBS
  .
  .  BASIC-ANWEISUNGEN DES ERSTEN BASIC-JOBS
  .
$ JOB-ENDE KARTE DES ERSTEN BASIC-JOBS
* JOB-KARTE DES ZWEITEN BASIC-JOBS
  .
$ JOB-ENDE KARTE

 .... WEITERE BASIC JOBS

<EOI>
```

Auf dem Plattenfile FEHLER stehen alle Fehlertexte, die vom FOSBIC-Compilersystem ausgegeben werden können. Ohne dieses Plattenfile wird nur die entsprechende Fehlernummer angegeben. Das Plattenfile INFO enthält Meldungen für die FOSBIC-Benutzer und orientiert sie über eventuelle Änderungen. Durch die CCL-Anweisung
ATTACH, TAPE 6= BENBIBL/M = W.
wird eine Benutzerbibliothek mit dem Namen BENBIBL angesprochen, auf der der Anwender seine BASIC-Programme archivieren kann.[75]

FOSBIC-Dialogaufruf, CD Cyber 174

Sobald sich das Computersystem nach Drücken der RETURN-Taste des Terminals (§) meldet, sind die nachstehenden Anweisungen einzugeben.

```
020202,XXXX
IAF
CHARGE,020202,020202
ATTACH,FOSBIC
ATTACH,TAPE4=FEHLER
ATTACH,TAPE5=INFO
ATTACH,TAPE6=BENBIBL/M=W
FOSBIC

... DANACH MELDET SICH DAS FOSBIC-SYSTEM
```

Danach meldet sich das FOSBIC-Compilersystem.
Im Gegensatz zum Monologbetrieb fehlt hier die BANNER-Anweisung, die nur eine Kennzeichnung der Ausgabeliste entsprechend dem angegebenen Text vornimmt. Die verwendete Dateiform der Benutzerbibliothek (BENBIBL) ist insofern von geringerer Bedeutung, als die gesamte interne Bibliotheksverwaltung durch das FOSBIC-Compilersystem vorgenommen wird.

75 Vgl. Bd. 2, Abs. 11.

Für BASIC-Kursteilnehmer am Fachbereich Wirtschaftswissenschaft der Justus-Liebig-Universität Giessen ist der gesamte Aufruf des FOSBIC-Compilersystems in einem kurzen maschinenabhängigen Systemprogramm zusammengefasst, so dass lediglich die beiden CCL-Anweisungen

```
GET,PROCFIL/UN=150210.
BEGIN,FOSBIC,PROCFIL,PROGINP=BENBIBL,DRUCK=WIWI.
```

sowohl im Monolog- als auch im Dialogbetrieb anzugeben sind. Die USER- und CHARGE-Anweisungen bleiben weiterhin notwendig.

14 BASIC-Standardisierung

141 ANSI-, ECMA- und ISO-Standards

Die Problematik verschiedener BASIC-Formen wurde in den USA insbesondere in Verbindung mit dem von Hochschulen – teilweise mit Unterstützung der National Science Foundation – gepflegten Softwareaustausch deutlich. Bei Verwendung nicht universell verfügbarer BASIC-Statements mussten die zum Austausch freigegebenen BASIC-Programme bei Übernahme auf andere Computersysteme oft wesentlich modifiziert werden, so dass die mit dem interuniversitären Softwareaustausch angestrebte Vermeidung der Duplikation von Entwicklungsarbeiten und Programmierkosten teilweise illusorisch gemacht wurde.
Diese Situation führte Gerald L. Isaacs – als Mitarbeiter des American College Testing Program – 1972 unter Verzicht auf Vollausschöpfung mancher BASIC-Dialekte zur Konzeption von ACT Transport BASIC, eines bewusst auf „interdialect translatability" ausgerichteten BASIC-Standards.[76]
ACT Transport BASIC wurde in der Folge – beispielsweise im Rahmen des Project CONDUIT[77] – weitgehend beachtet und regte zu weiteren Standardisierungsbemühungen an. Verwiesen sei insbesondere auf die Arbeiten von Stephen A. Garland (Dartmouth College), Gordon M. Bull (Hatfield Polytechnic) und W. Freeman (University of York), die auch ANSI Minimal BASIC weitgehend mitzugestalten vermochten.[78]

76 Gerald L. Isaacs. Interdialect Translatability of the BASIC Programming Language. Iowa City 1972.
77 Vgl. Trinka Dunnagan. CONDUIT Technical Transport Guidelines. Iowa City 1973, p. 26–33. – Vgl. Abs. 111.
78 Stephen J. Garland. Dartmouth BASIC: A Specification. Hanover 1973; G. M. Bull, W. Freeman and S. J. Garland. Specification for Standard BASIC. Manchester 1973.

Vom American National Standards Institute (ANSI) wurde 1974 die Kommission X3J2 zur Entwicklung von BASIC Standards konstituiert. Als Resultat der von der Kommission X3J2 äusserst intensiv vorangetriebenen Arbeiten konnte 1978 der

– American National Standard for Minimal BASIC

veröffentlicht werden.[79]

Seit 1974 befasste sich auch die European Computer Manufactures Association (ECMA) mit der BASIC-Standardisierung. Die Arbeiten der ECMA BASIC-Kommission TC21 führten 1978 zur Veröffentlichung von

– Standard ECMA-55 Minimal BASIC.[80]

ECMA Minimal BASIC stimmt – mit wenigen Ausnahmen – auch verbal mit ANSI Minimal BASIC überein.

Die beiden vorerwähnten Standards wurden in der Folge – mit geringen redaktionellen Änderungen – auch von der International Organization for Standardization (ISO) übernommen; verwiesen sei auf den

– Draft Proposal for Minimal BASIC

aus dem Jahre 1979.[81]

Die ANSI/ECMA-Standards für Minimal BASIC können im wesentlichen wie folgt zusammengefasst werden:

- BASIC ist als eine zeilenorientierte Programmiersprache zu konzipieren, die mindestens die folgenden Statements umfassen soll: DATA, DEF, DIM, END, FOR, GOSUB, GOTO, IF-THEN, INPUT, LET, NEXT, ON-GOTO, OPTION BASE, PRINT, RANDOMIZE, READ, REM, RESTORE, RETURN, STOP.
- Mit Hilfe der vorerwähnten Statements sollen numerische und alphanumerische Konstante und Variable verarbeitet werden können, wobei numerische Werte in den Grenzen 1E−38 bis 1E+38 und Strings mit 0 bis 18 Zeichen darstellbar sein sollen.
- Als Operationszeichen sind + (Addition), − (Subtraktion), * (Multiplikation), / (Division) und ^ (Exponentiation, involution) vorgesehen. Alternative Zeichen für die Exponentiation (** oder ↑) sind nicht zulässig.
- Als Standardfunktionen (implementation-supplied functions) sind

79 ANSI, American National Standards Institute. American National Standard for Minimal BASIC. New York 1978 – Im folgenden zitiert als ANSI Minimal BASIC.

80 ECMA, European Computer Manufactures Association. Standard ECMA-55 Minimal BASIC. January 1978. Geneva 1978.

81 ISO, International Organization for Standardization. Second Draft Proposal ISO/DP 6373. Minimal BASIC. Geneva 1979.

vorzusehen: ABS, ATN, COS, EXP, INT, LOG, RND, SGN, SIN, SQR, TAN, wobei RND ohne Argumentangabe erscheint.

- Bezüglich der einzelnen Statements gilt bei Minimal BASIC vor allem folgendes:

  INPUT-Statement: Dateneingabe soll formfrei erfolgen können.

  DATA-Statement: Es soll nur eine Gesamtdatei geschaffen werden, ohne Differenzierung zwischen numerischen und alphanumerischen Daten.

  LET-Statement: Es soll auch das implizite let-Statement zulässig sein; mehrfache Wertzuweisungen über ein LET-Statement sind nicht vorzusehen.

  IF-Statement: Als Vergleichsoperatoren sind < (kleiner), < = (kleiner/gleich), = (gleich), > (grösser), > = (grösser/gleich) und < > (nicht gleich) vorzusehen.

  Alphanumerische Vergleiche sind auf dem ASCII-Code basierend durchzuführen.

  ON-GOTO-Statement: Die zur Spezifikation der Sprungstelle verwendeten Werte arithmetischer Ausdrücke werden gerundet; falls keine der angegebenen Zeilennummern anzuspringen ist, erfolgt Fehlermeldung und Programmabbruch.

  PRINT-Statement: Zur Drucksteuerung sollen Komma, Strichpunkt und TAB-Funktion verfügbar sein; normalerweise wird die gesamte Druckbreite in fünf gleichbreite Druckzonen eingeteilt.

  DIM-Statement: Unter- und Obergrenze der Indizes sollen angegeben werden können; falls keine Untergrenze angegeben wird, ist diese mit Null anzunehmen.

  DEF-Statement: Mehrzeilige Benutzerfunktionen sind nicht vorgesehen.

  RANDOMIZE-Statement: Wahl zufälliger Startbasen für Zufallszahlengenerator; Argumentangabe ist nicht notwendig.

- Die Standards für Minimal BASIC sollen nicht nur für den Timesharing-Betrieb (interactive mode), sondern auch für Stapelbetrieb (batch mode) gelten.

ANSI Minimal BASIC wurde in die Tabellen 2 bis 16 eingearbeitet und als ANSI-Prozessor dargestellt. Ein Vergleich dieses Ideal-Compilers mit einer Auswahl von Real-Compilern zeigt, dass ANSI Minimal BASIC bereits als weitgehend realisiert gelten kann.[82]

82 Vgl. auch: Karl Weber. BASIC Programmier-Richtlinien für Austauschprogramme. Bern 1979.

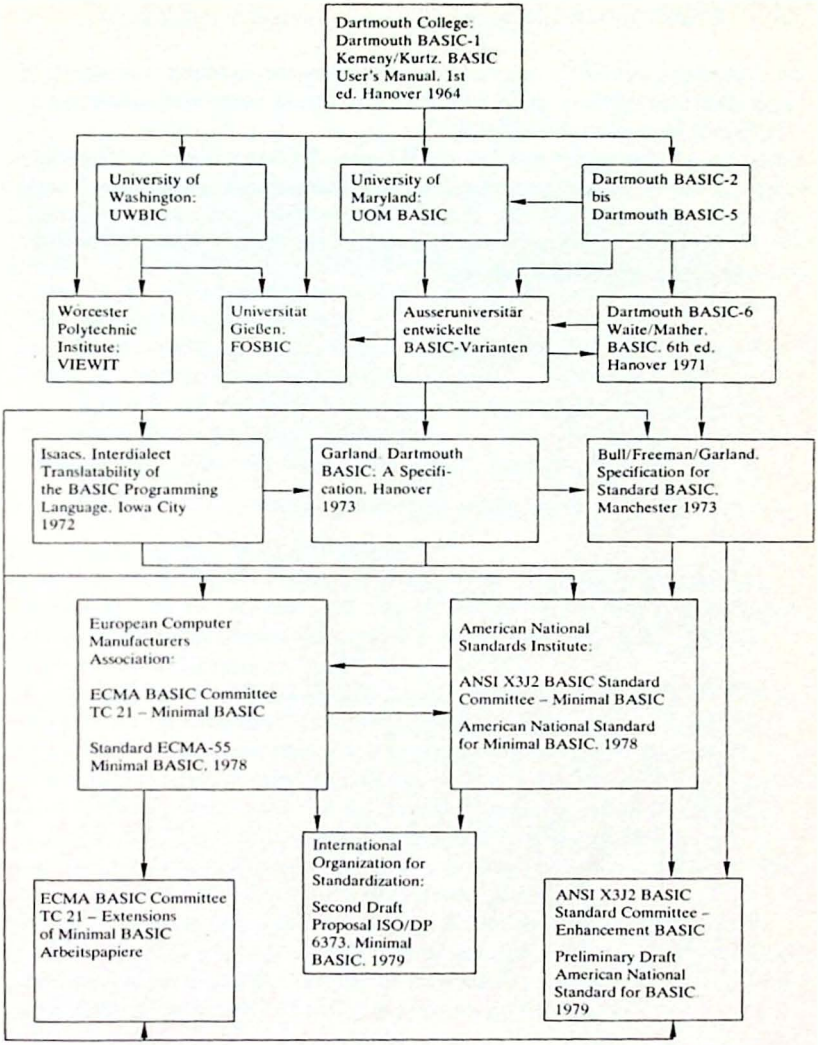


Abbildung 3. BASIC-Diversifikation und -Standardisierung

Die Arbeiten der ANSI- und ECMA-Kommissionen wurden nach 1978 fortgesetzt und führten 1979 zum Entwurf eines amerikanischen
– National Standard for BASIC.[83]
In diesem Arbeitspapier werden zur Hauptsache Vorschläge zur Standardisierung der BASIC-Sonderanweisungen gemacht, teilweise aber auch Erweiterungen bezüglich der BASIC-Elementaranweisungen angeregt. Die ANSI BASIC-Standards wurden im Rahmen dieser Arbeit in weitgehendem Masse mitberücksichtigt.[84]

83 ANSI, American National Standards Institute. American National Standard for BASIC. Preliminary Draft. September 1979. Hanover 1979. – Im folgenden zitiert als ANSI BASIC.
84 Vgl. auch Bd. 2, Abs. 14.

2 BASIC-Problembearbeitung

21 Problembeschreibung und Problemanalyse

Computergestützte Problemarbeit erfolgt in vier Arbeitsstufen. Die erste bezieht sich auf die Problembeschreibung und -analyse, wobei auch die zu verwendenden Lösungsmethoden und die für sie adäquaten Programmiersprachen festzulegen sind.

Die zur Bearbeitung anstehenden Probleme können einen unterschiedlichen Komplexitätsgrad aufweisen. Einfache Probleme liegen günstigstenfalls bereits formalisiert vor und lassen sich relativ leicht in programmierbare Form bringen. Verwiesen sei auf das bereits angeführte Problem der Lösung einer Gleichung zweiten Grades,[1] mit $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. Entsprechendes gilt aber auch für Matrizen- und andere arithmetische Operationen.[2]

Komplexe (Real-)Probleme sind zunächst in strukturell vereinfachte Formalprobleme umzusetzen, die im Hinblick auf computergestützte Auswertung möglicherweise – etwa zur erleichterten Algorithmisierung – weiter zu modifizieren sind.

Die Programmiersprache BASIC wurde – ähnlich wie FORTRAN und ALGOL, zu denen BASIC ursprünglich hinführen sollte – zunächst insbesondere zur Bearbeitung mathematisch-statistischer Probleme verwendet. Der gegen Ende der siebziger Jahre verstärkt zunehmende Einsatz als Programmiersprache auf Minicomputersystemen verlangte auch ausreichende Textverarbeitungsmöglichkeiten, die zunächst vor allem durch erweiterte alphanumerische Ausdrücke realisiert und in der Folge durch ANSI BASIC Standards abgedeckt wurden.[3]

Bezüglich der Bearbeitung mathematisch-statistischer Probleme bestehen bei Verwendung der Programmiersprache BASIC im wesentlichen nur maschinenbedingte Grenzen; Einzelheiten sind aus den Publikationen der Computer-Herstellerfirmen zu entnehmen.[4]

1 Vgl. Abs. 111.
2 Vgl. Abb. 8.
3 ANSI BASIC, p. 34–41.
4 Vgl. Abs. 112, 122. – Verwiesen sei auch auf: Gerald L. Isaacs. BASIC Revisited. An Update to Interdialect Translatability of the BASIC Programming Language. Iowa City 1976 und Molly Hepler. Memorandum to Holders of BASIC Revisited. Iowa City 1977.

22 Programmplanung

Die Programmplanung besteht im wesentlichen aus zwei Phasen und beinhaltet die Erstellung von Datenfluss- und Programmablaufplänen, wobei die erste Phase bei Bearbeitung einfacher Probleme in Wegfall kommen kann.

221 Datenflusspläne

Nach DIN 44300 stellt der Datenfluss die „zeitlichen Beziehungen im Ablauf einer Menge zusammengehöriger Vorgänge an Daten und Datenträgern dar".[5]
Die den Datenfluss betreffenden Vorgänge werden in entscheidendem Masse durch die Ablauforganisation, beim Einsatz von Datenverarbeitungsanlagen insbesondere durch den Programmablauf, geprägt.
Der Datenflussplan ist eine symbolische Darstellung dieser Ablauforganisation, zumeist unter Angabe der zeitlichen Folge der Einzelprogramme.
Datenflusspläne bestehen im wesentlichen aus Sinnbildern mit zugehörigem Text und orientierenden Verbindungslinien;[6] sie zeigen im wesentlichen die an einem Programmablauf beteiligten Datenträger bzw. Speicher und die beteiligten Geräte bzw. die von diesen Geräten ausgeführten Arbeiten.
Die Sinnbilder für Datenflusspläne gemäss DIN 66001 entsprechen weitgehend den in den USA üblichen ANSI-Symbolen (American National Standards Institute). Die wichtigsten Symbole für Datenträger bzw. Speicher sind aus Abbildung 4 ersichtlich. Weitere, für die Lochkarten-Datenverarbeitung relevante, Sinnbilder wurden für die Operationen Mischen, Trennen, Mischen mit gleichzeitigem Trennen und Sortieren entwickelt.[7]
Unter Verwendung dieser Symbole lassen sich auch komplexe Datenverarbeitungsprozesse konzis darstellen. Ein allgemein verständliches Beispiel ist aus Abbildung 5 ersichtlich.

5 Fachnormenausschuss Informationsverarbeitung (FNI) im Deutschen Normenausschuss (DNA). DIN 44300: Informationsverarbeitung, Begriffe. Berlin 1972, S. 8.
6 Fachnormenausschuss Informationsverarbeitung (FNI) im Deutschen Normenausschuss (DNA). DIN 66001: Informationsverarbeitung, Sinnbilder für Datenfluss- und Programmablaufpläne. Berlin 1966.
7 Über die seit 1970 gültigen ANSI-Symbole vgl. Marylin Bohl. Flowcharting Techniques. Chicago 1971, p. 131–144.

222 Programmablaufpläne

Programmablaufpläne sind konventionelle Zeichnungen von Programmen und Prozessen zur Veranschaulichung der Struktur und allgemeinen Operationsfolge eines Programms.
Bei umfangreichen EDV-Programmen ist es üblich, neben groben Programmablaufplänen noch detailliertere – oft auch als Blockdiagramme

DIN/ANSI-Sinnbild	Benennung
	Bearbeitung (processing)
	Lochkarte (punched card)
	Lochstreifen (perforated tape)
	Magnetband (magnetic tape)
	Sichtgerät (display)
	Fernschreiber (online keyboard)
	Schriftstück (document)
	Datenübertragung (communication link)
	Magnetplattenspeicher (ANSI Einheitssymbol für: disk, drum, random access)

Abbildung 4. Sinnbilder für Datenflusspläne

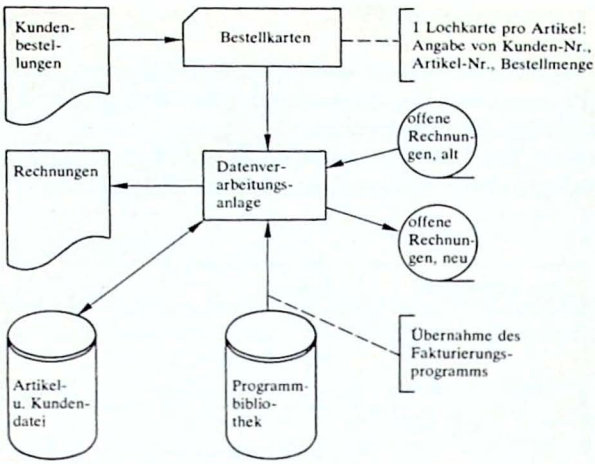


Abbildung 5. Fakturierung

bezeichnete – Pläne zu entwickeln, wobei die letzteren in der Regel alle zur Codierung[8] notwendigen Einzelheiten explizite angeben.

Zur Darstellung des Programmablaufs werden zweckmässigerweise die aus Abbildung 6 ersichtlichen Sinnbilder verwendet. Sie entsprechen nicht vollumfänglich den DIN/ANSI-Standards. So wird gemäss Abbildung 6 mit besonderen Ein- und Ausgabesymbolen (anstelle der DIN-Raute) gearbeitet und weiterhin eine Differenzierung zwischen IF- und FOR/NEXT-Verzweigungen vorgenommen.[9]

Die FOR/NEXT-Verzweigung wird zur Darstellung zyklischer Programmabläufe verwendet und ist in besonderem Masse zur BASIC-nahen Programmierung geeignet.

Zyklische Programmabläufe sind dadurch gekennzeichnet, dass eine Reihe von Befehlen zur Lösung einer Aufgabe mehrmals nacheinander durchlaufen wird. Es bestehen Programmschleifen, die im Programmablaufplan in der aus Abbildung 7 ersichtlichen Weise darzustellen sind.

8 Vgl. Abs. 231.

9 Vgl. hierzu auch: James W. Estes and B. Robert Ellis. Elements of Computer Science. New York 1973; Alexandra I. Forsythe, Thomas A. Keenan, Elliott I. Organick and Warren Stenberg. Computer Science: A First Course. New York 1969; Alexandra I. Forsythe, Thomas A. Keenan, Elliott I. Organick and Warren Stenberg. Computer Science: BASIC Language Programming. New York 1970; Thomas J. Schriber. Fundamentals of Flowcharting. New York 1969.

Abbildung 6. BASIC-orientierte Sinnbilder für Programmablaufpläne mit zugeordneten BASIC-Statements

Sinnbild	Benennung	Statement	Band, Abs.
	Operation, allgemein (ohne Eingabe/ Ausgabe)	LET let MAT LET MAT let	1, 332.31 1, 332.32 2, 331.21 2, 331.22
	Eingabe, Lochkarte Terminal	READ INPUT RESTORE MAT READ MAT INPUT	1, 332.13 1, 332.14 1, 332.15 2, 331.11 2, 331.12
	Ausgabe	PRINT PRINT TAB PRINT USING PAGE MAT PRINT MAT PRINT USING	1, 332.21 1, 332.22 1, 332.23 1, 332.25 2, 331.31 2, 331.32
	Verzweigung	IF/GOTO IF/THEN GOTO/ON GOTO/OF ON/GOTO OF/GOTO	1, 332.42 1, 332.42 1, 332.412 1, 332.412 1, 332.412 1, 332.412
	Programmschleife	FOR/NEXT LOOP/EXIT LOOP	1, 332.43 1, 332.44
	Unterprogramm	GOSUB RETURN GOSUB/ON GOSUB/OF ON/GOSUB OF/GOSUB	1, 332.47 1, 332.48 1, 332.47 1, 332.47 1, 332.47 1, 332.47
	Grenzstelle	STOP END	1, 332.45 1, 332.46
	Ein-/Ausgabe, sequentielle Files (Magnetband) Ein-/Ausgabe, index-sequentielle Files (Magnetplatte)	READ * WRITE * MAT READ * MAT WRITE * RESET * PRINT * INPUT * MAT PRINT * MAT INPUT * RESET *	2, 332.411 2, 332.311 2, 332.412 2, 332.312 2, 332.313/413 2, 332.321 2, 332.421 2, 332.322 2, 332.422 2, 332.323/423
	Übergangs-stelle	GOTO	1, 332.411
	Bemerkung	REM REMARK null COMMENT DIM BASE DEF DATA data IMAGE OPEN * CLOSE * BUILD * KILL *	1, 331.11 1, 331.11 1, 331.12 1, 331.13 1, 331.21 1, 331.22 1, 331.23 1, 332.11 1, 332.12 1, 332.23 2, 332.22 2, 332.23 2, 332.21 2, 332.24

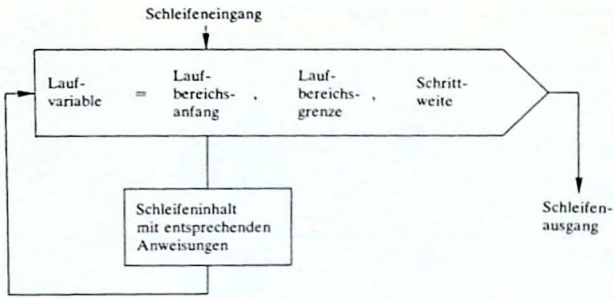
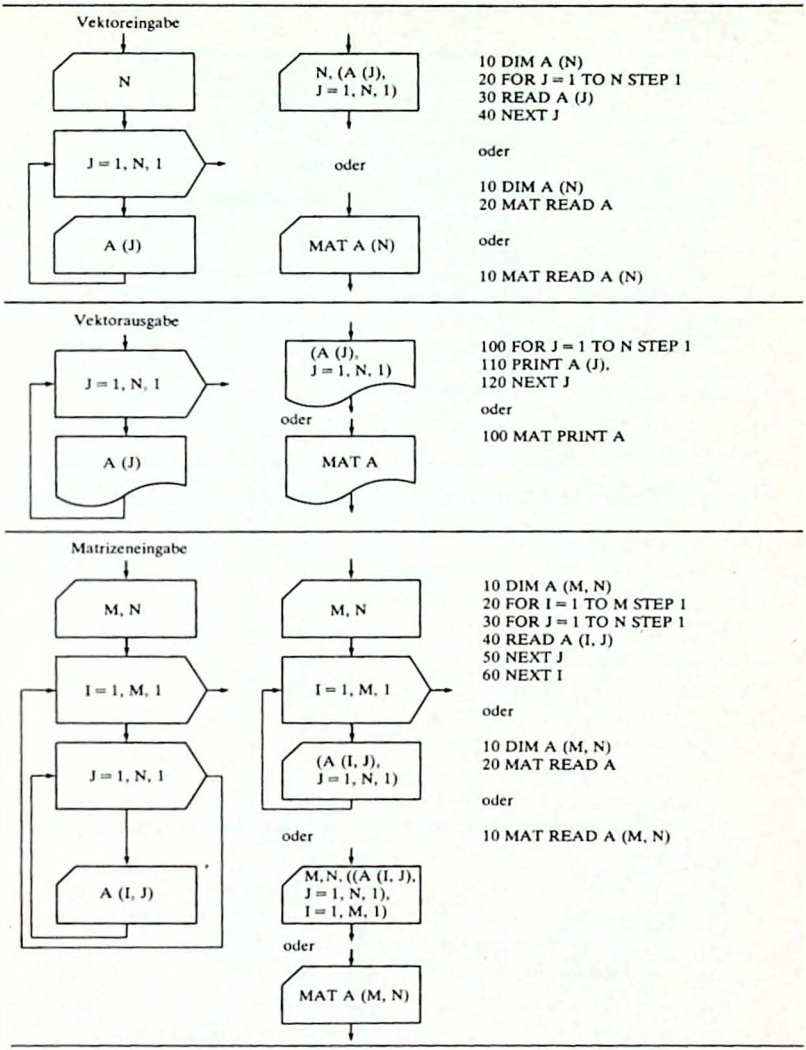
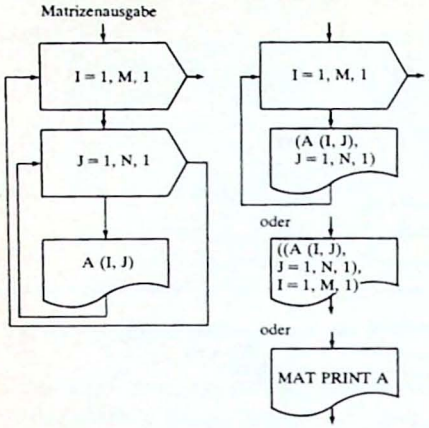


Abbildung 7. Schleifendarstellung mit FOR/NEXT-Symbol

Falls die Schrittweite gleich Eins ist, kann auf deren explizite Angabe verzichtet werden. Vermerkt sei auch, dass in einem Problem mehrere, ineinander verschachtelte Schleifen auftreten können, von denen jede durch eine entsprechende FOR/NEXT-Verzweigung darzustellen ist.[10] Besonders zu beachten ist, dass die Struktur der gewählten Programmiersprache möglicherweise auch die zu erstellenden Programmablaufpläne zu beeinflussen vermag. Dies trifft für BASIC in besonderem Masse bezüglich der Matrizenoperationen zu; diese sind – im Vergleich zu FORTRAN und ALGOL – besonders einfach programmierbar und werden in den Programmablaufplänen vorzugsweise nach den aus Abbildung 8 ersichtlichen Kurzdarstellungen ausgewiesen.
Wie sehr die vereinfachte Schleifendarstellung eine Entlastung der Programmablaufpläne zu bewirken vermag, ist aus Abbildung 9 (Sortierprogramm) und Abbildung 10 (Matrizenmultiplikation) beispielhaft ersichtlich. Die beigegebenen BASIC-Programme lassen auch die Auswirkung unterschiedlicher Sprachausschöpfung in der Programmablaufphase auf die Programmstruktur (Codierung) deutlich werden.
Im Gegensatz zu zyklischen Programmen weisen rein sequentielle Programme generell eine wesentlich einfachere Struktur auf. Sie sind dadurch gekennzeichnet, dass die zur Lösung einer Aufgabe erforderlichen Befehle nur einmal durchlaufen werden; es treten mithin – vgl. Abbildung 11 – keine Programmschleifen auf.
Bezüglich der Zeichentechnik hat sich in der Praxis die vertikale Darstellung weitgehend durchzusetzen vermocht; in der amerikanischen Literatur wird – etwa im Standardwerk von Thomas J. Schriber – allerdings auch mit horizontal verlaufenden Plänen (die sich besonders platzsparend darstellen lassen) gearbeitet.[11]

10 Vgl. Abs. 332.43.
11 Vgl. Abb. 12.


Vektoreingabe
N
J = 1, N, 1
A (J)
N, (A (J),
J = 1, N, 1)
oder
MAT A (N)
10 DIM A (N)
20 FOR J = 1 TO N STEP 1
30 READ A (J)
40 NEXT J
oder
10 DIM A (N)
20 MAT READ A
oder
10 MAT READ A (N)
Vektorausgabe
J = 1, N, 1
A (J)
(A (J),
J = 1, N, 1)
oder
MAT A
100 FOR J = 1 TO N STEP 1
110 PRINT A (J),
120 NEXT J
oder
100 MAT PRINT A
Matrizeneingabe
M, N
I = 1, M, 1
J = 1, N, 1
A (I, J)
M, N
I = 1, M, 1
(A (I, J),
J = 1, N, 1)
oder
M, N, ((A (I, J),
J = 1, N, 1),
I = 1, M, 1)
oder
MAT A (M, N)
10 DIM A (M, N)
20 FOR I = 1 TO M STEP 1
30 FOR J = 1 TO N STEP 1
40 READ A (I, J)
50 NEXT J
60 NEXT I
oder
10 DIM A (M, N)
20 MAT READ A
oder
10 MAT READ A (M, N)




```
100 FOR I = 1 TO M STEP 1
110 FOR J = 1 TO N STEP 1
120 PRINT A (I, J)
130 NEXT J
140 NEXT I
```

oder

```
100 MAT PRINT A
```

Abbildung 8. Vereinfachte Schleifendarstellung für Dateneingabe und -Ausgabe

```
 10  REM SORTIERPROGRAMM *NUMERISCH*
 20  DIM A(100)
 30  READ N
 40  FOR J=1 TO N
 50    READ A(J)
 60  NEXT J
 70  FOR I=1 TO N-1
 80    FOR J=I+1 TO N
 90      IF A(J) > A(I) GOTO 139
100      LET T=A(I)
110      LET A(I)=A(J)
120      LET A(J)=T
130    NEXT J
140 NEXT I
150 FOR J=1 TO N
160   PRINT A(J),
170 NEXT J
180 STOP
190 DATA 9,9,8,7,6,5,4,3,2,1
200 END

*** EVERYTHING SEEMS OK -- LET'S GO AHEAD
  1            2            3            4            5
  6            7            8            9
```

```
 10  REM SORTIERPROGRAMM *NUMERISCH*
 20  DIM A(100)
 30  READ N
 40  MAT READ A(N)
 50  FOR I=1 TO N-1
 60    FOR J=I+1 TO N
 70      IF A(J) > A(I) GOTO 110
 80      LET T=A(I)
 90      LET A(I)=A(J)
100      LET A(J)=T
110    NEXT J
120 NEXT I
130 MAT PRINT A(N)
150 STOP
160 DATA 9,9,8,7,6,5,4,3,2,1
170 END
```

BASIC-Programme zu Abbildung 9.

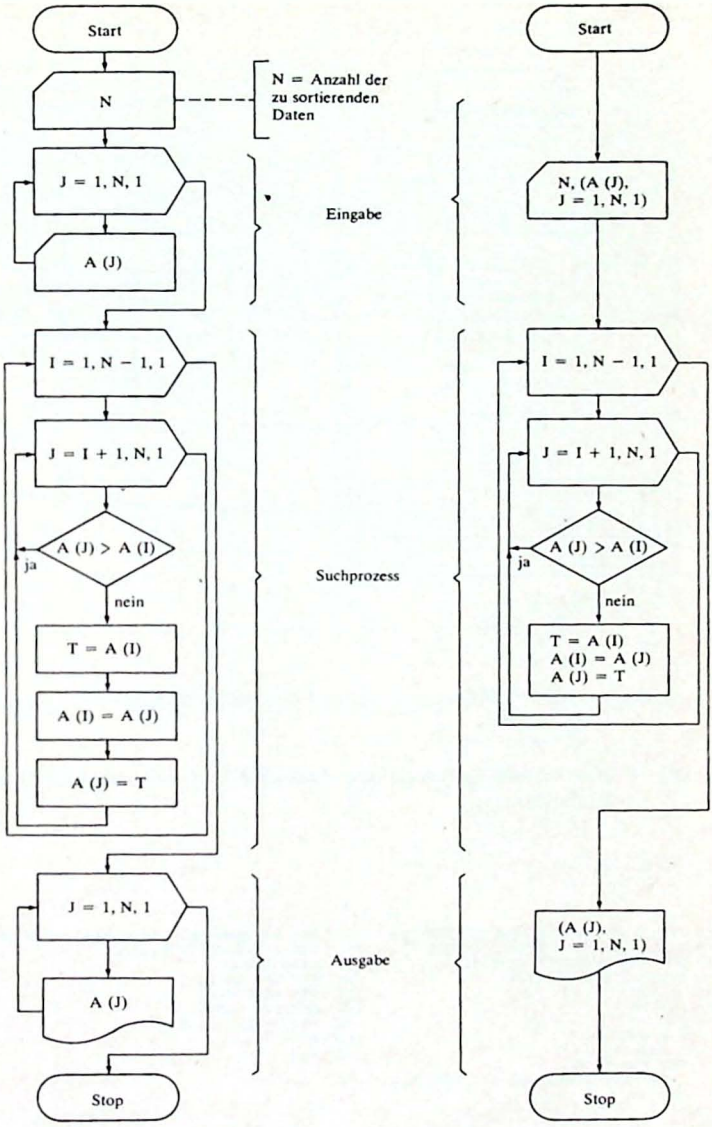
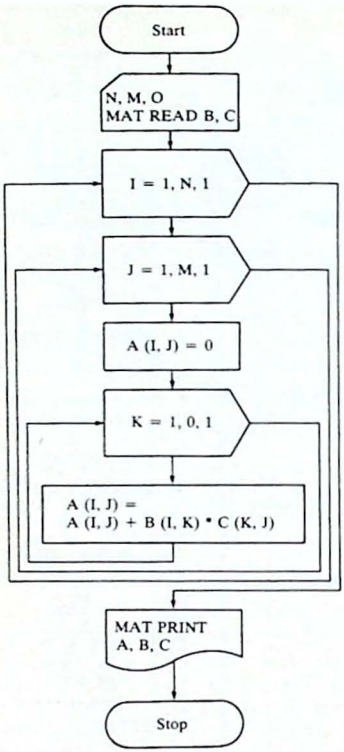


Abbildung 9. Sortierprogramm mit expliziter und vereinfachter Schleifendarstellung für Dateneingabe und -Ausgabe



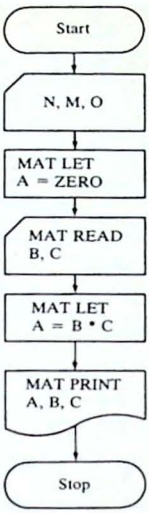


Abbildung 10. Matrizenmultiplikation mit expliziter und vereinfachter Schleifendarstellung

```
10  REM EXPLIZITE MATRIZENMULTIPLIKATION
20  REM DIMENSIONIERUNGSREGEL LAUTET
30  REM DIM A(N,M),B(N,O),C(O,M)
40  DIM A(3,4),B(3,7),C(7,4)
50  READ N,M,O
60  MAT READ B,C
70  FOR I=1 TO N
80    FOR J=1 TO M
90    LET A(I,J)=0
100     FOR K=1 TO O
110       LET A(I,J)=A(I,J)+B(I,K)*C(K,J)
120     NEXT K
130   NEXT J
140 NEXT I
150 MAT PRINT A,B,C
160 DATA .......
170 END
```

```
10  REM VEREINFACHTE MATRIZENMULTIPLIKATION
40  DIM A(3,4),B(3,7),C(7,4)
50  READ N,M,O
60  MAT LET A=ZER
70  MAT READ B,C
80  MAT LET A=B*C
90  MAT PRINT A,B,C
100 DATA ....
110 END
```

BASIC-Programme zu Abbildung 10.

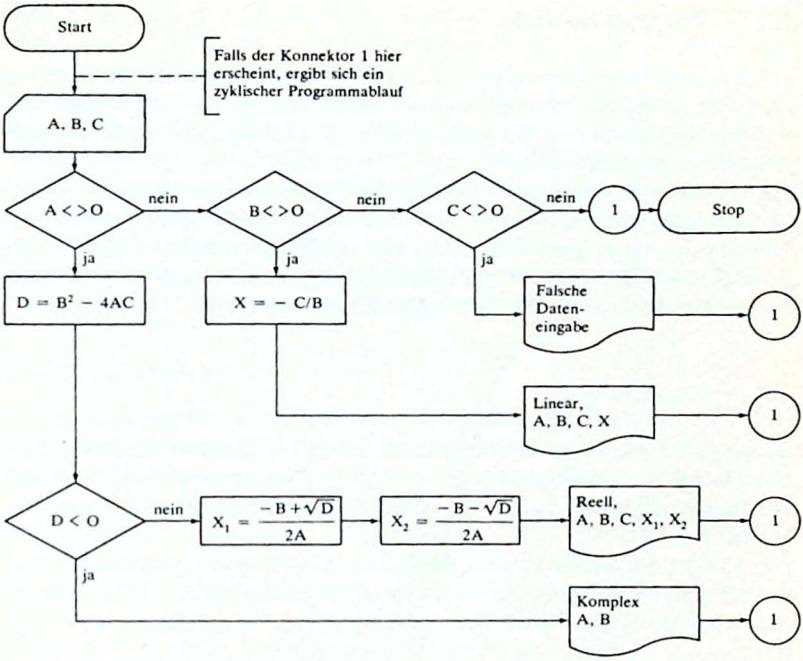


Abbildung 11. Gleichung zweiten Grades, sequentieller Programmablauf

23 Programmierung

Der Stufe der Programmplanung folgt die Programmierung. Sie umfasst die Erarbeitung der zur computergestützten Lösung eines Problems notwendigen Folge von Anweisungen, die in sich geschlossen als Computerprogramm bezeichnet wird.[12]
Die Erstellung operationaler Computerprogramme erweist sich als relativ aufwendiger, im allgemeinen mehrfach zu durchlaufender Prozess, der üblicherweise aus den Phasen Codierung, Programmübertragung, Programmumwandlung und Programmtest besteht. Bei besonders einfachen Problemen kann die erste und letzte Phase wegfallen.

231 Codierung

Bezogen auf die Programmiersprache BASIC beinhaltet Codierung die Umsetzung der erstellten Programmablaufpläne in eine Folge BASIC-spezifischer Anweisungen (Programmsätze, Statements) und deren Niederschrift.
Zur Codierung werden vorzugsweise sprachbezogen strukturierte Vordrucke[13] verwendet, von denen die Programmsätze leicht auf Lochkarten zu übernehmen bzw. über Terminals weiter zu verarbeiten sind.

232 Programmübertragung

Bei der Programmübertragung muss grundsätzlich zwischen den beiden Arbeitsweisen Monolog- und Dialogbetrieb unterschieden werden.

232.1 Monologbetrieb

Beim BASIC-Monologbetrieb[14] erfolgt die Programmübertragung in der Regel auf Lochkarten, wobei sich BASIC insofern als zeilenorientierte Programmiersprache (line oriented language) erweist,[15] als für jeden Programmsatz – dargestellt in einer Zeile des Codierformulars – eine einzige Lochkarte zu verwenden ist.

12 "A program is a sequence of statement lines, the last of which contains an end-statement and each of which is identified by a keyword." ANSI Minimal BASIC, p. 11.
13 Vgl. Abs. 326.3.
14 Vgl. Abs. 13.
15 "BASIC is a line-oriented language." ANSI Minimal BASIC, p. 11.

Ein vollumfänglich in Lochkarten übernommenes Programm kann ein relativ umfangreiches Kartenpaket ausmachen; es wird als Quellenprogramm (source program) bezeichnet.
Charakteristisch für den Monologbetrieb ist es, dass normalerweise die Statementnummernfolge[15a] der einzelnen BASIC-Anweisungen keine vorgegebene Ordnung besitzen muss. Beim FOSBIC-Compilersystem ist es sogar möglich, im Monologbetrieb nur diejenigen Anweisungen mit einer Statementnummer zu versehen, auf die eine Programmverzweigungsanweisung[15b] explizit Bezug nimmt.
Zu beachten ist, dass manche BASIC-Programmiersysteme auch im Monologbetrieb eine aufsteigende Statementnumerierung verlangen.

232.2 Dialogbetrieb

Im Minicomputerbereich wird nahezu ausschliesslich der BASIC-Dialogbetrieb bevorzugt.[15c] Die Programmierung erfolgt hierbei zeilenweise vom Codierungsformular über ein Dialogterminal direkt in ein Computersystem. Die dabei erzeugten BASIC-Quellenprogramme werden sofort als Dateien auf externen Datenträgern – bevorzugt wird dabei die Magnetplatte – abgespeichert. Mittels des BASIC-Programmiersystems[15d] können diese BASIC-Programme in einer Prorammbibliothek verwaltet und gewartet werden, so dass alle notwendigen Änderungen ebenfalls über ein Dialogterminal eingegeben werden.
Bezeichnend für die dialogorientierte Programmübertragung ist die fortlaufende und zwingend notwendige Statementnumerierung. Die Statementnummer dient im Dialogprogramm nicht nur als Sprungziel für ein GOTO-Statement[15e], sondern auch als Ordnungskriterium für die Anweisungsfolge innerhalb des BASIC-Programms. Im Dialogbetrieb werden daher alle Anweisungen stets in aufsteigender Reihenfolge nach ihren Statementnummern geordnet, unabhängig von der Eingabefolge in der Programmübertragung. Werden zwei oder mehr Anweisungen mit derselben Statementnummer eingegeben, so überschreibt die zuletzt übertragene Anweisung immer die vorausgehende. Wird lediglich eine Statementnummer ohne Statementinhalt[15f] angegeben und existiert bereits eine BASIC-Anweisung, die diese Statementnummer trägt, so wird die gesamte Anweisung einschliesslich Statementnummer aus der Programmliste gelöscht.

15a Vgl. Abs. 326.2.
15b Vgl. Abs. 332.4.
15c Vgl. Bd. 2, Abs. 12.
15d Vgl. Bd. 2, Abs. 11.
15e Vgl. Abs. 332.41.
15f Vgl. Abs. 326.1.

Das nachfolgende Beispiel zeigt die Wirkungsweise der Statementnummern bei der Programmübernahme im Dialogbetrieb.

```
10 PRINT "DIES IST ";
20 PRUNT
30 PRINT "MEIN ";
40 PRINT "TEST"
50 END
20
40 PRINT "TESTPROGRAMM IN BASIC"
35 PRINT "ERSTES ";
LIST

10 PRINT "DIES IST ";
30 PRINT "MEIN ";
35 PRINT "ERSTES ";
40 PRINT "TESTPROGRAMM IN BASIC"
50 END
```

Weitere Einzelheiten zur Verwaltung von Dialogprogrammen werden im Zusammenhang mit dem BASIC-Programmiersystem erläutert.[15g]

233 Programmumwandlung

des BASIC-Quellenprogramm ist – über den verfügbaren BASIC-ompiler – computerintern in Maschinensprache zu übertragen, wobei ieser Umwandlungsprozess ein- oder mehrstufig erfolgen kann. Programmwandlungsziel ist die Generation eines lauffähigen Objektprogramms (object program).
Bei syntaktischen Fehlern im Quellenprogramm werden – in Abhängigkeit von der Benutzerfreundlichkeit des BASIC-Compilers[16] – mehr oder weniger ausführlich gehaltene Fehlerprotokolle ausgedruckt, aufgrund deren das Quellenprogramm und vielleicht auch der ihm zugrundeliegende Programmablaufplan zu überarbeiten sind. Erst wenn keine syntaktischen Fehler mehr erkannt werden, fällt als Resultat des Programmumwandlungsprozesses ein lauffähiges Objektprogramm an.
Das Objektprogramm wird üblicherweise nicht – etwa in Form eines Binärdecks – an den Benutzer abgegeben; es bleibt lediglich zur unmittelbaren Verarbeitung der in Verbindung mit dem Quellenprogramm eingegebenen Daten computerintern gespeichert und wird nach Programmablauf wieder gelöscht. Dies trifft auch bei Verwendung der UWBIC- und FOSBIC-Compiler zu.[17]

15g Vgl. Bd. 2, Abb. 11.
16 Für den UWBIC-Compiler vgl. Abs. 121.
17 Vgl. Abs. 122.2.

233.1 Monologbetrieb

Grundsätzlich unterscheidet sich das im Monologbetrieb erzeugte Objektprogramm nicht von dem eines im Dialogbetrieb erstellten. Lediglich beim Auftreten von Fehlern besteht beim Monologbetrieb für den BASIC-Anwender keine Möglichkeit zur sofortigen Korrektur. Dies führt bei umfangreichen Programmen dazu, dass meist nur durch eine Reihe von Programmumwandlungstests die syntaktischen BASIC-Programmfehler korrigiert werden können.

233.2 Dialogbetrieb

Im Dialogbetrieb ergeben sich für den BASIC-Anwender bei der Programmumwandlung erhebliche Vorteile, die vor allem bei benutzerfreundlichen BASIC-Prozessoren deutlich hervortreten und die Anzahl der Programmumwandlungstests gegenüber dem Monologbetrieb stark reduzieren.
Im BASIC-Dialogbetrieb wird bereits während der interaktiven Programmübertragung der einzelnen BASIC-Anweisungen über das Dialogterminal eine syntaktische Überprüfung vorgenommen, so dass zumindest die Mehrzahl der formalen BASIC-Fehler sofort erkannt, dem Benutzer mitgeteilt und von diesem sofort korrigiert werden kann. Meistens ist dazu lediglich eine Neueingabe der zuletzt niedergeschriebenen BASIC-Anweisung in korrigierter Form notwendig. Mit Beendigung der Programmübertragung ist damit meistens auch gleichzeitig ein syntaktisch fehlerfreies BASIC-Programm erstellt.
Die für die Programmumwandlung auf den einzelnen Computersystemen installierten BASIC-Compiler oder BASIC-Interpreter[17a] unterscheiden sich allerdings erheblich in ihrer syntaktischen Prüfgenauigkeit während der Programmumwandlung. Vor allem BASIC-Interpreter sind teilweise nicht in der Lage, syntaktisch komplexere BASIC-Fehler festzustellen. Zum Beispiel fällt es einem BASIC-Interpreter nicht auf, wenn das NEXT-Statement vor dem dazugehörigen FOR-Statement eingegeben wird oder wenn zu einem FOR-Statement das dazugehörige NEXT-Statement fehlt.[17b]
Je umfangreicher die syntaktisch formale Prüfung des erstellten BASIC-Programms während der Programmumwandlung ist, desto geringer wird

17a Zum Begriff des BASIC-Compilers und BASIC-Interpreters vgl. Weber und Türschmann, FOSBIC-Compiler, S. 33–34.
17b Zum FOR/NEXT-Statement vgl. Abs. 332.43.

die Anzahl der notwendigen Programmtests, da sich der BASIC-Benutzer bei diesen Tests im wesentlichen nur noch auf die logisch richtige Ablauffähigkeit der erzeugten Programmstruktur konzentrieren muss.

234 Programmtest

Das Objektprogramm muss auch auf logische Richtigkeit überprüft werden, was zweckmässigerweise anhand computerunabhängig nachvollziehbarer Testaufgaben erfolgt.
Nach eventuellen Fehlerberichtigungen kann aufgrund des Quellenprogramms ein voll operationales Objektprogramm generiert und zur Bearbeitung von Nutzaufgaben eingesetzt werden.
Falls das Objektprogramm Bestandteil eines komplexeren Programmsystems ist, müssen die Systemverbindungen durch Kettentests geprüft und Koppelungsschwierigkeiten durch Umprogrammierung überwunden werden; stellen sich beim Kettentest schwerwiegende Fehler im Gesamtsystem heraus, drängt sich möglicherweise ein Neuansatz bei der Programmplanung auf. Die dabei wiederum zu durchlaufenden Arbeitsstufen sind aus Abbildung 12 ersichtlich.
Auch beim Programmtest erweist sich der Dialogbetrieb zunächst von Vorteil, da eventuell auftretende logische Programmfehler sofort über das Dialogterminal korrigiert werden können.
Bei sehr komplexen und umfangreichen BASIC-Programmen bietet jedoch der Dialogbildschirm eine geringere Übersichtlichkeit als eine ausgedruckte Programmliste. Aus diesem Grund erscheint es vorteilhaft, dass einige BASIC-Compiler auch im Alternativbetrieb einsetzbar sind, wodurch umfangreiche BASIC-Programme im Monologbetrieb getestet und im Dialogbetrieb korrigiert werden können.
Die FOSBIC-Version 2 bietet dem Anwender die letzterwähnte Möglichkeit, indem durch entsprechende BASIC-Systembefehle (LIST-PRINT bzw. RUNPRINT) die BASIC-Programmliste bzw. die gesamte vom BASIC-Programm erzeugte Ausgabe auch auf dem Schnelldrucker gelistet werden kann. Damit besteht auf einfache Weise die Möglichkeit, eine sogenannte "Hardcopy" einer Terminalsitzung zu erzeugen.[17c]

24 Programmdokumentation

Jede Problembearbeitung sollte – auch in einfachsten Fällen – mit der Programmdokumentation zum Abschluss kommen.

17c Vgl. Bd. 2, Abs. 112.3.

Bei der Dokumentation von BASIC-Programmen kann im wesentlichen den bisher erwähnten Arbeitsschritten gefolgt werden, wobei auf adäquate Problembeschreibung und sorgfältige Erläuterung der Ein- und Ausgabedaten mit entsprechenden Beispielen besonderer Wert zu legen ist. Unbedingt erforderlich ist ein vollständiger Ausdruck (listing) des Gesamtprogramms, aus dem der Umfang der BASIC-Sprachausnutzung – und damit der Grad der unmittelbaren Übertragbarkeit des Programms auf andere Anlagen – feststellbar ist. Wertvoll wären – bei grösseren Programmen – auch Hinweise auf die Integrationsmöglichkeit in übergeordnete Programmsysteme, beispielsweise in eine Methoden- oder Modellbank.

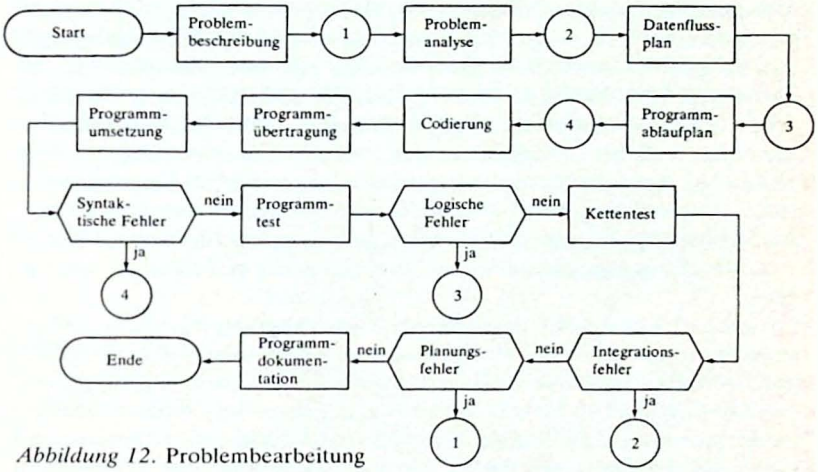


Abbildung 12. Problembearbeitung

25 BASIC-Sprachausschöpfung

Die Grundstruktur eines BASIC-Programms kann – worauf bei Behandlung der Schleifenverarbeitung kurz hingewiesen wurde[18] – durch den Grad der BASIC-Sprachausschöpfung in entscheidendem Masse beeinflusst werden. Der zu realisierende Sprachausnutzungsgrad ist deshalb generell und/oder problembezogen mit grosser Umsicht festzulegen, wobei die folgenden Grundsätze zu beachten sind.
Programmieranfänger sollten sich bezüglich des zu aktivierenden Sprachumfangs generell auf ANSI Minimal BASIC beschränken, wobei zu-

18 Vgl. Abb. 8.

nächst sogar mit Vorteil auf den Einbezug weniger gebräuchlicher Statements bzw. Statementformen verzichtet werden kann. So scheint es durchaus vertretbar, bezüglich der Dateneingabe und -Ausgabe anfänglich nur mit den – bei BASIC besonders einfachen – READ- und PRINT-Statements zu arbeiten und erst bei Bedarf auch Spezialanweisungen (RESTORE, PRINT TAB, PRINT USING) zu übernehmen.
Enhanced BASIC wird den Benutzern allgemein – compilerbedingt – nur in unterschiedlichem Umfang zur Verfügung stehen. Eine möglichst weitgehende Sprachausschöpfung ist für fortgeschrittene Programmierer bezüglich der nicht zum Austausch bestimmten Programme anzustreben. Andererseits drängt sich hinsichtlich der – beispielsweise interuniversitär – einem weiteren Benutzerkreis verfügbar zu machenden Programme bei Verwendung von Spezialstatements relativ grosse Zurückhaltung auf. Compilerspezifische oder nur wenig verbreitete Sprachausprägungen sollten generell vermieden werden; dies gilt beispielsweise für die Statements FOR/WHILE, FOR/UNTIL, IF/UNLESS und IF/WHILE. Auch auf Programmverkettungsanweisungen sollte in Austauschpro-rammen tunlichst verzichtet werden; bei den Fileanweisungen drängt ch in bezug auf die Statements zur index-sequentiellen Dateiverarbei-ıng Zurückhaltung auf.[19] Jedenfalls sind Spezialanweisungen in Aus-auschprogrammen sinnvoll zu kommentieren, um nachfolgenden Benutzern die Programmadaptation in weitestgehendem Masse zu erleichtern.[20]
Generell wird eine befriedigende Sprachausschöpfung nur unter Einhaltung der – zur Zeit noch in Entwicklung begriffenen – ANSI/ECMA Standards für Enhanced BASIC erreichbar sein.[21] Durch Anpassung der BASIC-Compiler an diese Standards kann seitens der Computerhersteller ein gewichtiger Beitrag zur Software-Austauschbarkeit geleistet und damit auf der Ebene der BASIC-Benutzer wesentlich zur Vermeidung von Doppelarbeit[22] beigetragen werden.

19 Vgl. Bd. 2, Abs. 332.4. – Beachtenswert ist auch die Feststellung von Garland L. Isaacs: "File handling is not directly translatable and the program writer should attend carefully to file input and file output statements when designing translatable programs." Isaacs, Interdialect Translatability of the BASIC Programming Language, p. 14.

20 Vgl. hierzu: Karl Weber. BASIC Programmier-Richtlinien für Austauschprogramme. Bern 1979.

21 Vgl. Bd. 2, Abs. 14.

22 Über die geringe Austauschbarkeit amerikanischer Software (1970 generell nur 2%) vgl. Penny Barbe. Techniques for Automatic Program Translation. In: J. T. Tou (ed.). Software Engineering Coins III. Vol. 1. New York 1970, p. 151.

3 BASIC-Struktur

31 Formalisierte Sprachdarstellung

Programmiersprachen können grundsätzlich nur durch übergeordnete (Meta-)Sprachen präzise dargestellt werden, wobei die zur Sprachbeschreibung dienenden Symbole der übergeordneten Sprache (Metazeichen) in der zu betrachtenden Sprache nicht mehr auftreten dürfen und nur zu deren Erläuterung dienen.[1]
Zur metalinguistischen Sprachdarstellung wird üblicherweise auf die Backus Normal Form bzw. Backus-Naur Form (BNF) zurückgegriffen, deren Metazeichen und Leseart aus Tabelle 1 ersichtlich ist.[2]

Zeichen	Bedeutung	zu lesen als
: =	Definitionszeichen	besteht aus oder ist definiert durch
⟨ ⟩	Beinhaltung (metalinguistische Klammern)	
Nebeneinanderstellung ohne metaling. Zeichen	Verkettung	gefolgt durch
|	ODER-Zeichen	oder

Tabelle 1. Metalinguistische BNF-Zeichen

Das Definitionszeichen (equivalence) tritt grundsätzlich bei allen Sprachdefinitionen auf. Weiterhin ist die *links* vom Definitionszeichen auftretende Einheit immer durch ein metalinguistisches Klammerpaar (angular brackets) zu umfassen.
Rechts des Definitionszeichens erscheinen in den primären Sprachdefinitionen die verfügbaren Grundsymbole; sie sind nicht in Klammern auszu-

1 Vgl. B. Higman. Programmiersprachen. Eine vergleichende Studie. München 1972, S. 17.
2 Jean E. Sammet. Programming Languages: History and Fundamentals. Englewood Cliffs 1969, p. 52–57.

weisen. So ist etwa der gesamte BASIC-Ziffernvorrat durch die metalinguistische Gleichung

⟨Ziffer⟩ : = 0|1|2|3|4|5|6|7|8|9

darzustellen.
Metalinguistische Klammern treten rechts des Definitionszeichens dann auf, wenn die zu definierende Einheit (syntactic unit) nicht unmittelbar und ausschliesslich auf Grundsymbole zurückgeführt werden kann. Da die Programmiersprache BASIC neben den vorerwähnten Ziffern auch Alpha- und Sonderzeichen als Grundsymbole verwendet, gilt:

⟨Grundsymbol⟩ : = ⟨Ziffer⟩ | ⟨Alphazeichen⟩ | ⟨Sonderzeichen⟩

In der vorstehenden Gleichung wird auch das metalinguistische Oder-Zeichen (or) verwendet.
Dass zur metalinguistischen Verkettungsdarstellung (concatenation) die Nebeneinanderstellung (juxtaposition) von bereits definierten Sprachheiten genügt, zeigt die folgende Gleichung:

⟨Ganzzahl⟩ : = ⟨Ziffer⟩ | ⟨Ganzzahl⟩ ⟨Ziffer⟩

Die BNF-gestützte Sprachdarstellung wird in der BASIC-Literatur relativ häufig – in ausgeweiteter Form auch zur Darstellung von ANSI Minimal BASIC – verwendet.[3] Sie liegt auch den folgenden Ausführungen zugrunde, wobei zur besseren Erfassung der Sprachzusammenhänge alle Definitionen numeriert und deren unmittelbare systembezogenen Relationen vermerkt werden.
Die *linke* Zahl gibt die Nummer der Definitionsgleichung an, in der die Einheit definiert ist. Die Kopplung der Definitionsgleichungen untereinander wird durch die *rechts* stehende(n) Zahl(en) aufgezeigt.

32 Definitionen

321 Grundsymbole

Die im folgenden zu behandelnden BASIC-Grundsymbole sind aus Abbildung 13 ersichtlich.

3 ANSI, American National Standards Institute. American National Standard for Minimal BASIC. New York 1978. – Im folgenden zitiert als ANSI Minimal BASIC.

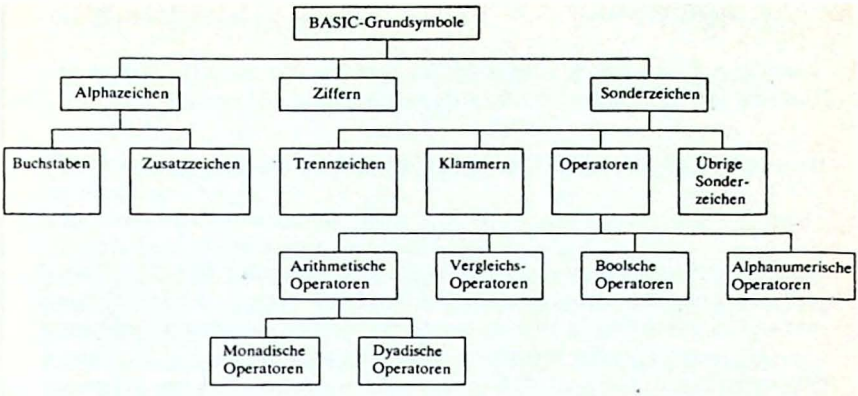


Abbildung 13.

Repräsentanten aller BASIC-Grundsymboltypen, für die generell die Basisgleichung

$$\underset{1}{\langle \text{BASIC-Grundsymbol} \rangle} := \underset{2}{\langle \text{Alphazeichen} \rangle} \mid \underset{5}{\langle \text{Ziffer} \rangle} \mid \underset{6}{\langle \text{Sonderzeichen} \rangle}$$

gilt, finden sich im folgenden BASIC-Programm.

```
10 READ A
20 DATA 1
30 PRINT ≠DIES IST MEIN ≠,A,≠.VERSUCH≠
40 PRINT ≠EIN PROGRAMM ZUM LAUFEN ZU BRINGEN≠
50 END
```

Das angegebene Programm ergibt – bei Verwendung des FOSBIC-Compilers – das Druckbild

```
DIES IST MEIN          1       .VERSUCH
EIN PROGRAMM ZUM LAUFEN ZU BRINGEN
```

Auch die übrigen in den Text aufgenommenen BASIC-Programme wurden FOSBIC-konform strukturiert. Dem Leser wird empfohlen, sämtliche Programme übungsweise zum Laufen zu bringen.

321.1 Alphazeichen

Unter der Bezeichnung Alphazeichen werden die BASIC-konformen Buchstaben und Zusatzzeichen zusammengefasst. Allgemein gilt:

$$\underset{2}{\langle \text{Alphazeichen} \rangle} \underset{1}{:=} \underset{3}{\langle \text{Buchstabe} \rangle} \mid \underset{4}{\langle \text{Zusatzzeichen} \rangle}$$

Aus der vorstehenden Gleichung ergibt sich, dass der Begriffsinhalt 2 (Alphazeichen) bereits in der Bestimmungsgleichung 1 (BASIC-Grundsymbol) in Erscheinung tritt; andererseits werden die Begriffselemente über die nachfolgenden Bestimmungsgleichungen 3 (Buchstabe) und 4 (Zusatzzeichen) definiert. Diese Darstellungsweise wird im folgenden generell beibehalten.
Während nach ANSI Minimal BASIC nur die Grossbuchstaben zugelassen sind, berücksichtigt dessen Weiterentwicklung zu ANSI BASIC[3a] auch Kleinbuchstaben, so dass für die Buchstaben nunmehr

$$\underset{3}{\langle \text{Buchstabe} \rangle} \underset{2}{:} = \underset{3a}{\langle \text{Grossbuchstabe} \rangle} \mid \underset{3b}{\langle \text{Kleinbuchstabe} \rangle}$$

gilt.

Für die Menge der BASIC-konformen Grossbuchstaben gilt die metalinguistische Gleichung

$$\underset{3a}{\langle \text{Grossbuchstabe} \rangle} \underset{3}{:=} \text{A}|\text{B}|\text{C}|\text{D}|\text{E}|\text{F}|\text{G}|\text{H}|\text{I}|\text{J}|\text{K}|\text{L}|\text{M}|\text{N}|\text{O}|\text{P}|\text{Q}|\text{R}|\text{S}|\text{T}|\text{U}|\text{V}|\text{W}|\text{X}|\text{Y}|\text{Z}$$

Alle BASIC-Compiler lassen die Buchstaben A bis Z des lateinischen Alphabets zu. ANSI Minimal BASIC entsprechend sind die Buchstaben nach dem ASCII-Code (American National Standard Code for Information Interchange) darzustellen.
Die Grossbuchstaben werden bei allen BASIC-Compilern ausser in Stringinhalten[3b] zur Bildung von Variablen-, Feld- und Funktionsnamen[3c] verwendet.

3a ANSI BASIC, p. 9.
3b Vgl. Abschn. 322.2.
3c Vgl. Abschn. 323.11, 323.21, 323.12, 323.22, 325.12.

Für die Kleinbuchstaben gilt:

⟨Kleinbuchstabe⟩ : = a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|
3b 3 v|w|x|y|z|

Kleinbuchstaben werden, auch nach ANSI BASIC[3d], nur in Textinhalten verwendet.
Die Zusatzzeichen umfassen jene Kategorie von darstellbaren Zeichen, die als Sonderzeichen keine besondere Funktion erhalten, aber zumindest in Stringinhalten auftreten können. Die Verwendung dieser Zeichen zur Bildung von Variablen-, Feld- und Funktionsnamen ist auch nach ANSI BASIC nicht erlaubt.
Die Zusatzzeichen enthalten meist sprachspezifische Zeichenerweiterungen, wie zum Beispiel die im Deutschen bekannten Umlaute Ä, Ö und Ü.[4]

⟨Zusatzzeichen⟩ : = Ä|Ö|Ü|ä|ö|ü|$|à|#|§§§
4 2

In der metalinguistischen Gleichung steht §§§ für weitere compilerspezifisch erlaubte Zusatzzeichen, wie zum Beispiel für die 64 graphischen Zusatzzeichen des PET 2001.[5]
Der FOSBIC-Compiler verfügt über keine Zusatzzeichen, so dass – ANSI Minimal BASIC entsprechend – die Beziehung
⟨Alphazeichen⟩ := ⟨Buchstabe⟩
zutrifft. Dagegen sind nach der FOSBIC-Version 2 Kleinbuchstaben zulässig; sie dürfen allerdings nur in Texten (alphanumerischen Konstanten) vorkommen. Dasselbe gilt für alle Zusatzzeichen, die auf dem verwendeten Computersystem dargestellt werden können.

321.2 Ziffern

Sämtliche BASIC-Compiler lassen die arabischen Ziffern 0 bis 9 zu; es gilt mithin:

⟨Ziffer⟩ : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
5 1

3d ANSI BASIC, p. 18, 35, 73.
4 Nixdorf. System-Software Business-BASIC. Paderborn 1979, S. 3.24.
5 Commodore. Bedienungshandbuch PET 2001. Neu-Isenburg 1978, S. 2.

Nach ANSI Minimal BASIC sind die Ziffern (digits) im ASCII-Code darzustellen.[6]
Ziffern dienen primär zur Bildung von Zahlen (numbers); sie können aber auch in Zeichenfolgen (strings) verwendet werden oder zur Bildung von Variablennamen dienen.
Beispiele: 12 und 2.71828 sind aus Ziffern gebildete Zahlen; in '2.71828' oder 'E = 2.718' wurden Ziffern zur Bildung von Zeichenfolgen verwendet. In dem Variablennamen A3 erscheint ebenfalls eine Ziffer.[7]

321.3 Sonderzeichen

Gemäss Abbildung 13 gilt die Beziehung

$$\underset{6}{\langle\text{Sonderzeichen}\rangle} \underset{1}{:=} \left\langle \begin{matrix}\text{Trenn-}\\ \text{zeichen}\end{matrix} \right\rangle_{7} \Big| \underset{8}{\langle\text{Klammer}\rangle} \Big| \underset{9}{\langle\text{Operator}\rangle} \Big| \left\langle \begin{matrix}\text{Übriges}\\ \text{Sonder-}\\ \text{zeichen}\end{matrix} \right\rangle_{16}$$

mit

$$\underset{7}{\langle\text{Trennzeichen}\rangle} \underset{6}{:=} \,,\,|\,;\,|\,:=\,|\,\text{Leerzeichen}$$

Das Trennzeichen = tritt nur im LET-Statement auf und hat hier eine zuweisende Funktion; es ist von dem als Vergleichsoperator verwendeten Gleichheitszeichen im IF-Statement zu unterscheiden. Die maschineninterne Zeichendarstellung ist für beide Symbole identisch.
Leerzeichen (spaces, blanks) werden innerhalb einzelner BASIC-Statements benutzt, um deren Lesbarkeit zu verbessern. Dabei sind nach ANSI Minimal BASIC beispielsweise auch die Statementstrukturen

```
100 LET X = Y*Z+12
100 LET X = Y * Z + 12
100 LETX=Y*Z+12
```

zulässig. Die letzterwähnte Form erscheint allerdings wenig empfehlenswert; sie findet sich in grösserem Umfang nur in der älteren BASIC-Literatur.[8] Zu vermeiden sind auch Statementstrukturen in der Art von

```
1 00 LET X = Y*Z+12
```

6 ANSI Minimal BASIC, p. 10.
7 Vgl. Abs. 322.1, 322.2 und 323.11.
8 Vgl. etwa: Clifford H. Springer, Robert E. Herlihy, Robert T. Hall and Robert I. Beggs. Statistical Inference. Homewood 1966, p. 317–327.

100 L E T X = Y*Z+12 ,
da sie von ANSI Minimal BASIC nicht gedeckt werden.[9]
Der Einsatz der übrigen Trennzeichen ergibt sich aus der allgemeinen Struktur der Programmiersprache BASIC. So werden beispielsweise Komma und Semikolon u. a. auch zur Drucksteuerung und der Punkt bei der Darstellung von Gleitkommazahlen verwendet.[10] Der Doppelpunkt findet vor allem in Fileanweisungen[10a] und zur Teilzeichenkettenbegrenzung[10b] Verwendung.

$$\underset{8}{\langle \text{Klammer} \rangle} \underset{6}{:=} (\mid) \mid ' \mid '' \mid [\mid]$$

auch eckige Klammern; sie sind nach ANSI Minimal BASIC allerdings nicht vorgesehen und werden praktisch nur selten – in Fileanweisungen – verwendet.[11] Runde Klammern dienen insbesondere zur eindeutigen resp. BASIC-konformen Darstellung arithmetischer Ausdrücke. Anführungszeichen zur Stringbegrenzung.
So wird etwa der Ausdruck 3/ −4 nur in der Form 3/ (−4) den Anforderungen der Programmiersprache 'BASIC' "gerecht", womit auch gleichzeitig zwei alternativ verwendbare Stringdarstellungen aufgezeigt sind.[12]
Die Operatoren lassen sich gemäss Abbildung 13 in vier Gruppen einteilen, so dass

$$\underset{9}{\langle \text{Operator} \rangle} \underset{6}{:=} \underset{10}{\langle \text{Arithmetischer Operator} \rangle} \mid \underset{13 \mid 13a \mid 13b}{\langle \text{Vergleichs-Operator} \rangle} \mid \underset{15}{\langle \text{Boolscher Operator} \rangle} \mid \underset{14}{\langle \text{Alphanumerischer Operator} \rangle}$$

gilt, wobei sich die arithmetischen Operatoren noch einmal aufteilen.

$$\underset{10}{\langle \text{Arithmetischer Operator} \rangle} \underset{9}{:=} \underset{11}{\langle \text{Monadischer Operator} \rangle} \mid \underset{12}{\langle \text{Dyadischer Operator} \rangle}$$

Monadische Operatoren werden in der deutschen Literatur üblicherweise Vorzeichen genannt; in der anglo-amerikanischen Literatur ist die Bezeichnung „unary operator" geläufig. Es gilt:

9 ANSI Minimal BASIC, p. 33–34.
10 Vgl. Abs. 332.21 und 322.12.
10a Vgl. Bd. 2, Abs. 332.
10b Vgl. Abs. 323.23.
11 ANSI Minimal BASIC, p. 10; vgl. Bd. 2, Abs. 332.4.
12 Der FOSBIC-Compiler lässt beide Formen der Stringdarstellung zu.

⟨Monadischer Operator⟩ : = + | −
11 10

Plus und minus werden beispielsweise in −A und +3.14 sowie im BASIC-Statement 10 LET R1 = −R2 als monadische Operatoren verwendet.
Als Operationszeichen für Addition, Subtraktion, Multiplikation, Division und Exponentiation dienen die dyadischen Operatoren, in der anglo-amerikanischen Literatur „binary arithmetic operators" genannt.
Die Darstellung der dyadischen Operatoren ist in BASIC noch nicht standardisiert; allgemein gilt:

⟨Dyadischer Operator⟩ : = + | − | * | / | ** | ↑ | ^
12 10

Als Multiplikationszeichen wird in BASIC generell der Stern (*) verwendet. Als Exponentiationszeichen ist nach ANSI Minimal BASIC der Zirkumflex (^) vorgesehen; manche Compiler lassen aber auch Doppelstern **) oder Pfeil (↑) als Exponentiationszeichen zu.[13] Vgl. Tabelle 2 und as auf S. 14 angeführte BASIC-Programm.
Auch bezüglich der Darstellung der Vergleichsoperatoren bestehen compilerspezifische Unterschiede. Abgesehen von der nur bei wenigen Compilern zulässigen Verwendung von ≤ oder =<, ≥ oder => sowie ≠, wird mit

⟨Vergleichsoperator⟩ : = < | < = | = | > = | > | <>
13 9

ANSI Minimal BASIC zumeist eingehalten.[14] Vgl. Tabelle 2.
Bei einzelnen Compilern müssen bzw. können die Vergleichsoperatoren translitiert werden; diesfalls gilt:

⟨Vergleichsoperator⟩ : = LT | LE | EQ | GE | GT | NE
13a 9

Beim FOSBIC-Compiler sind alle Darstellungsformen der Vergleichsoperatoren zulässig, so dass durch

⟨Vergleichsoperator⟩ : = < | <= | = | >= | > | <> |
13b 9 LT | LE | EQ | GE | GT | NE |
=< | => | ≠

13 ANSI Minimal BASIC, p. 14, 35.
14 Sonderfälle bestehen; so verfügt etwa NEWBASIC (Abb. 112.9) über zusätzliche Vergleichsoperatoren.

eine optimale Kompatibilität für die Übernahme fremder BASIC-Programme gewährleistet wird.
Die alphanumerischen Operatoren unterliegen ebenfalls einer compilerspezifischen Darstellungsweise, indem allgemein

$$\left\langle \begin{matrix}\text{Alphanumerischer}\\ \text{Operator}\end{matrix} \right\rangle := \& \mid + \mid ,$$
14 9

gilt, wobei nach ANSI BASIC nur das Et-Zeichen (&, ampersand) zugelassen wird.[14a] Der FOSBIC-Compiler lässt in der Version 2 – in Übereinstimmung mit ANSI BASIC – & als alphanumerischen Operator zu.
Die Boolschen Operatoren $\cap$, $\cup$ und $\neg$ sind bei allen BASIC-Compilern – falls überhaupt – nur transliteriert verfügbar; mithin gilt:

⟨Boolscher Operator⟩ : = AND | OR | NOT | AND NOT | OR NOT
15 9

Bei Nichtverfügbarkeit Boolscher Operatoren wird die Leistungsfähigkeit der BASIC-Compiler praktisch nur unwesentlich eingeschränkt. So sind etwa die Statements
100 IF A GE B THEN 200
und
100 IF A>B OR A=B THEN 200
äquivalent. Beide Statements besagen, dass im Falle von A ⩾ B das Programm mit Statement 200, andernfalls aber mit dem nächstfolgenden BASIC-Satz fortzusetzen ist.
Unter komplexeren Verhältnissen kann mittels logischer Operatoren – ceteris paribus – verkürzt programmiert werden, wie das einzeilige Statement
100 IF A<B OR C=D PRINT A+B ELSE PRINT C, D
beispielsweise zeigt.
Beim FOSBIC-Compiler sind Boolsche Operatoren nicht verfügbar; sie sind auch in ANSI Minimal BASIC nicht vorgesehen.

14a ANSI BASIC, p. 37.

Prozessor	Dydadischer Operator EXP	Vergleichsoperatoren LT	LE	EQ	GE	GT	NE	Alphanumerische Operatoren
ANSI Minimal BASIC	^	<	<=	=	>=	>	<>	
Dartmouth BASIC-6	^	<	<= =<	=	>= =>	>	<> ><	&
UOM BASIC/ UBASIC	**	<	<= =<	=	>= =>	>	<> ><	&
IBM CALL BASIC	^ **	<	<= ≦	=	>= ≧	>	<> =	&
UWBIC	**	LT	LE	=	GE	GT	NE	
FOSBIC	**|^|↑ einstellbar	< LT	<= LE	= EQ	>= GE	> GT	<> NE	(Version 2)

EXP – Exponentiation, LT – kleiner als, LE – kleiner/gleich, EQ – gleich, GE – grösser/gleich, GT – grösser als, NE – nicht gleich. Von den angeführten BASIC-Compilern lässt nur UOM BASIC/UBASIC Boolsche Operationen zu.

Tabelle 2. Verfügbarkeit von BASIC-Operatoren

Die Zahl der übrigen BASIC-konformen Sonderzeichen ist compilerabhängig, so dass

$$\langle \text{Übriges Sonderzeichen} \rangle := \underset{16}{\%} \mid \underset{6}{\#} \mid \cdot \mid : \mid ! \mid ? \mid \S\S\S$$

gilt, wobei mit §§§ die zusätzlich verfügbaren Sonderzeichen dargestellt seien.

Die Sonderzeichen werden bevorzugt zur Kennzeichnung der IMAGE-Anweisung[15] verwendet. Das #-Zeichen ist typisch bei Fileanweisungen.[16] Anstelle des $-Zeichens zur Darstellung alphanumerischer Variabler findet man auch das englische £-Zeichen.[17]

15 Vgl. Abs. 332.23.
16 Vgl. Bd. 2, Abs. 332.4.
17 Typisch für ICL-BASIC, vgl. Abs. 112.7.

BASIC-Konstante sind während des Programmablaufs unverändert bleibende numerische und alphanumerische Grössen[18]; sie werden in der anglo-amerikanischen Literatur häufig als „numeric-constants" und „string-constants" bezeichnet[19], wobei sich die ersteren – vgl. Abbildung 14 – insbesondere nach der Art ihrer Formatierung weiter aufgliedern lassen[20].

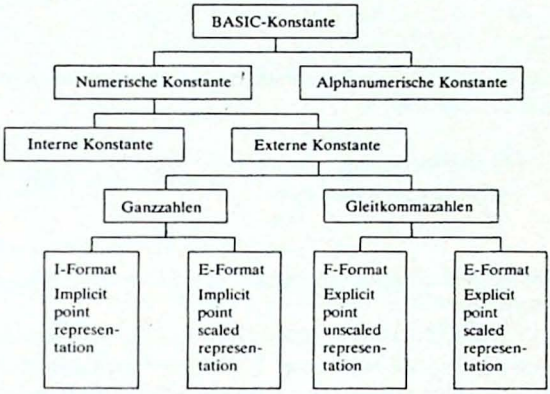


Abbildung 14.

Allgemein unterteilen sich die Konstanten in numerische und alphanumerische.

$$\underset{17}{\langle \text{Konstante} \rangle} := \underset{18}{\left\langle \begin{matrix} \text{Numerische} \\ \text{Konstante} \end{matrix} \right\rangle} \mid \underset{32}{\left\langle \begin{matrix} \text{Alphanumerische} \\ \text{Konstante} \end{matrix} \right\rangle}$$

18 "Constants can denote both scalar numeric values and string values." ANSI Minimal BASIC, p. 12.

19 Garland, Dartmouth BASIC, p. 16–17; Bull, Freeman and Garland, Specification for Standard BASIC, p. 52–56; ANSI Minimal BASIC, p. 12.

20 Vgl. auch ANSI Minimal BASIC (p. 12), wo zwischen "implicit point representation, implicit point scales representation, explicit point unscaled representation, explicit point scaled representation" unterschieden wird.

322.1 *Numerische Konstante*

Numerische Konstante (Zahlen) können bereits im Compilerprogramm – als interne Konstante[21] – enthalten sein; normaliter werden sie aber im Quellenprogramm festgelegt. Dementsprechend gilt:

$$\langle \text{Numerische Konstante} \rangle_{18} := \left\langle \begin{matrix}\text{Compilerinterne}\\ \text{numerische}\\ \text{Konstante}\end{matrix} \right\rangle_{19} \Bigg| \left\langle \begin{matrix}\text{Compilerexterne}\\ \text{numerische}\\ \text{Konstante}\end{matrix} \right\rangle_{20}$$

Relativ wenige Compiler verfügen über compilerinterne Konstante (internal constants), für die

$$\left\langle \begin{matrix}\text{Compilerinterne}\\ \text{numerische Konstante}\end{matrix} \right\rangle_{19,18} := \&PI \mid \&E \mid \&\&$$

utrifft, falls mit && compilerspezifisch zusätzlich verfügbare Festwerte argestellt werden.
Bei IBM CALL BASIC ist beispielsweise – ausser Festwerten für π und e – auch noch die interne Konstante $\sqrt{2}$ in der Form &SQR2 verfügbar.
Bezüglich der compilerexternen Konstanten wird in der Literatur oft zwischen Integer-Konstanten und Real-Konstanten unterschieden. Im folgenden werden die Bezeichnungen Ganz- und Gleitkommazahl bevorzugt, was auch in der metalinguistischen Gleichung

$$\left\langle \begin{matrix}\text{Compilerexterne}\\ \text{numerische Konstante}\end{matrix} \right\rangle_{20,18} := \langle \text{Ganzzahl} \rangle_{21} \mid \langle \text{Gleitkommazahl} \rangle_{26}$$

zum Ausdruck kommt.

322.11 Ganzzahlen

Ganzzahlen (Integer-Konstante) sind in I- oder E-Format dargestellte Ziffernfolgen ohne Dezimalpunkt. Es gilt mithin die Dichotomie

21 Nach ANSI Minimal BASIC (p. 12) und ANSI BASIC sind keine compilerinterne Konstanten vorgesehen.

$$\underset{21}{\langle \text{Ganzzahl} \rangle} \underset{20}{:=} \underset{22}{\left\langle \begin{matrix}\text{I-formatierte}\\ \text{Ganzzahl}\end{matrix} \right\rangle} \mid \underset{25}{\left\langle \begin{matrix}\text{E-formatierte}\\ \text{Ganzzahl}\end{matrix} \right\rangle}$$

I-formatierte Ganzzahlen sind reine Ziffernfolgen mit einem eventuell linksstehenden Vorzeichen. Die entsprechende metalinguistische Gleichung lautet:

$$\underset{22}{\left\langle \begin{matrix}\text{I-formatierte}\\ \text{Ganzzahl}\end{matrix} \right\rangle} \underset{21}{:=} \underset{23|24}{\left\langle \begin{matrix}\text{Vorzeichen-}\\ \text{lose Ganz-}\\ \text{zahl}\end{matrix} \right\rangle} \mid \underset{11}{\left\langle \begin{matrix}\text{Monadi-}\\ \text{scher}\\ \text{Operator}\end{matrix} \right\rangle} \underset{23|24}{\left\langle \begin{matrix}\text{Vorzeichen-}\\ \text{lose Ganz-}\\ \text{zahl}\end{matrix} \right\rangle}$$

mit

$$\underset{23}{\left\langle \begin{matrix}\text{Vorzeichen-}\\ \text{lose Ganzzahl}\end{matrix} \right\rangle} \underset{22}{:=} \underset{5}{\langle \text{Ziffer} \rangle} \mid \underset{23|24}{\left\langle \begin{matrix}\text{Vorzeichen-}\\ \text{lose Ganzzahl}\end{matrix} \right\rangle} \underset{5}{\langle \text{Ziffer} \rangle}$$

wofür bei expliziter Darstellung auch

$$\underset{24}{\left\langle \begin{matrix}\text{Vorzeichen-}\\ \text{lose Ganzzahl}\end{matrix} \right\rangle} \underset{22}{:=} \underset{5}{\langle \text{Ziffer} \rangle} \mid \underset{5}{\langle \text{Ziffer} \rangle} \underset{5}{\langle \text{Ziffer} \rangle} \mid \ldots$$

geschrieben werden könnte.
Der monadische Operator (Plus oder Minus) kann unmittelbar vor der ersten Ziffer stehen oder aber durch Leerzeichen (spaces) davon abgetrennt sein. Vorzeichenlose Ganzzahlen werden immer als positiv interpretiert, so dass bei Darstellung positiver Werte auf explizite Vorzeichenangabe verzichtet werden kann.
I-formatierte Ganzzahlen sind beispielsweise +89, + 12, 89, −21; über compilerspezifische Grenzwerte vgl. Tabelle 3.
E-formatierte Ganzzahlen bestehen aus einer I-formatierten Ganzzahl mit einem unmittelbar folgenden Exponententeil, der sich in den Buchstaben E und eine weitere I-formatierte Ganzzahl aufteilt. In metalinguistischer Schreibweise kann dies durch

$$\underset{25}{\left\langle \begin{matrix}\text{E-formatierte}\\ \text{Ganzzahl}\end{matrix} \right\rangle} \underset{21}{:=} \underset{22}{\left\langle \begin{matrix}\text{I-formatierte}\\ \text{Ganzzahl}\end{matrix} \right\rangle} \text{E} \underset{23}{\left\langle \begin{matrix}\text{vorzeichenlose}\\ \text{Ganzzahl}\end{matrix} \right\rangle} \mid$$

$$\underset{22}{\left\langle \begin{matrix}\text{I-formatierte}\\ \text{Ganzzahl}\end{matrix} \right\rangle} \text{E+} \underset{23}{\left\langle \begin{matrix}\text{vorzeichenlose}\\ \text{Ganzzahl}\end{matrix} \right\rangle}$$

dargestellt werden.

Die zur Darstellung einer E-formatierten Ganzzahl benötigte und dem Exponententeil vorgestellte I-formatierte Ganzzahl heisst Mantisse. Die im Exponententeil erscheinende vorzeichenlose Ganzzahl stellt den Exponenten zur Basis zehn dar. Über compilerspezifische Maximalwerte orientiert Tabelle 3.

Beispiele für E-formatierte Ganzzahlen sind −11E5, entsprechend −1100000 und +27E+3, entsprechend +27000.

In BASIC-Statements können – unter Beachtung der compiler- bzw. maschinenbedingten Grenzen – beliebig strukturierte Zahlen nebeneinander auftreten; beispielsweise enthält das Statement

200 LET A3 = 2E+10 − 31/B

eine E- und eine I-formatierte Ganzzahl.

Der FOSBIC-Compiler lässt E-formatierte Ganzzahlen nur in DATA-Statements zu.[22] Die Version 2 verarbeitet E-formatierte Ganzzahlen in allen Statements.

22 Vgl. Abs. 322.11.

Prozessor	Numerische Konstante						Alphanumerische Konstante
	I-Format Dezimalziffern in DATA-Statements oder per Terminal		F-Format Dezimalziffern in DATA-Statements oder per Terminal		E-Format maschineninterne Darstellung		maximale Zeichenzahl
	minimal	maximal	minimal	maximal	minimal	maximal	
ANSI Minimal BASIC	6		6		1E–38	1E+38	Zeilenlänge
Dartmouth BASIC-6		8		6 oder 7	1.46937E–39	1.70141E+38	120
UOM BASIC/ UBASIC		8		8	1E–39	1E+38	60
IBM CALL BASIC	7	16	6	15	1E–78	1E+75	18
UWBIC	maschinenabhängig; intern besteht keine Abprüfung auf einen Zahlen „Underflow" oder „Overflow"; es erfolgt ein Programmabbruch durch das Maschinensystem.						75 aufgelöst in Gruppen zu je 5 Zeichen
FOSBIC allgemein	einstellbar		einstellbar		einstellbar		75 aufgelöst Gruppen zu je 5 Zeichen
CD 3300 Doppelwort-option	± 140737488355327		11 signifikante Ziffern		1E–308	1E+307	

Tabelle 3. Grenzwerte von Konstanten

322.12 Gleitkommazahlen

Gleitkommazahlen (Real-Konstante) weisen generell einen Dezimalpunkt auf. Sie können F- oder E-formatiert dargestellt werden, so dass

$$\underset{26}{\langle \text{Gleitkommazahl} \rangle} \underset{20}{:=} \underset{27}{\langle \text{F-formatierte Gleitkommazahl} \rangle} \mid \underset{30}{\langle \text{E-formatierte Gleitkommazahl} \rangle}$$

gilt.
Für F-formatierte Gleitkommazahlen ergeben sich insofern etwas komplexe Bildungsregeln, als der Dezimalpunkt auch als erstes oder letztes Zeichen erscheinen kann; einzelne Compiler drucken bei Gleitkommazahlen mit Werten kleiner als 1 allerdings immer eine führende Null.

$$\underset{27}{\langle \text{F-formatierte Gleitkommazahl} \rangle} \underset{26}{:=} \underset{28}{\langle \text{Vorzeichenlose F-formatierte Gleitkommazahl} \rangle} \mid \underset{11}{\langle \text{Monadischer Operator} \rangle} \underset{28}{\langle \text{Vorzeichenlose F-formatierte Gleitkommazahl} \rangle}$$

$$\underset{28}{\langle \text{Vorzeichenlose F-formatierte Gleitkommazahl} \rangle} \underset{27}{:=} \underset{23}{\langle \text{Vorzeichenlose Ganzzahl} \rangle} \cdot \underset{23}{\langle \text{Vorzeichenlose Ganzzahl} \rangle} \mid$$

$$\underset{23}{\langle \text{Vorzeichenlose Ganzzahl} \rangle} \cdot \mid \cdot \underset{23}{\langle \text{Vorzeichenlose Ganzzahl} \rangle}$$

Beispiele für F-formatierte Gleitkommazahlen sind 0.25, + 12.34, − 128.16, .333 und 148. Zu beachten ist, dass – amerikanischer Praxis folgend – ausschliesslich mit Dezimalpunkt zu arbeiten ist.
E-formatierte Gleitkommazahlen weisen einen Exponententeil der Form

$$\underset{29}{\langle \text{Exponententeil} \rangle} \underset{30}{:=} \text{E} \; \underset{22}{\langle \text{I-formatierte Ganzzahl} \rangle}$$

auf und sind im übrigen wie folgt strukturiert:

$$\underset{30}{\langle \text{E-formatierte Gleitkommazahl} \rangle} \underset{26}{:=} \underset{31}{\langle \text{Mantisse} \rangle} \; \underset{29}{\langle \text{Exponententeil} \rangle}$$

⟨Mantisse⟩ : = ⟨F-formatierte Gleitkommazahl⟩
31 30 27

Die gezeigte Darstellungsform wird auch als halblogarithmische Schreibweise bezeichnet. Mantisse und Exponent können positiv oder negativ sein. Positive Vorzeichen müssen bei Eingabedaten nicht geschrieben werden und bleiben im allgemeinen auch bei der Datenausgabe unterdrückt.
E-formatierte Gleitkommazahlen empfehlen sich besonders zur Darstellung sehr grosser und sehr kleiner Zahlen. Beispiele:
1.23E−6=0.00000123,
0.1234E+3=0.1234E3=123.4,
1.776E5=1.776E+05=177600.
Ein einfaches, FOSBIC-konformes BASIC-Statement könnte beispielsweise wie folgt lauten:
20 PRINT 11.67, 12.3E10
Vermerkt sei, dass nicht alle Compiler in sämtlichen Statements E-formatierte Gleitkommazahlen verarbeiten.
Nichteinhaltung der compilerspezifischen Formatierungsregeln wird durch entsprechende Fehlermeldungen angezeigt.
Beim FOSBIC-Compiler sind maximal dreistellige Exponentenwerte zulässig. Werden Daten, deren Exponent den eingestellten maximal zulässigen Wert überschreitet, eingelesen, so erfolgt die Meldung

```
          I CAN ONLY HANDLE NUMBERS BETWEEN 10** 99 AND 10**(- 99)
SEE BASIC TEXTBOOK  A 322.12
```

Überschreitet der Exponent den eingestellten Maximalwert während des Programmablaufs, so erfolgt die Meldung

```
PROGRAM STOPPED -- I CAN HANDLE ONLY NUMBERS BETWEEN 10** 99 AND 10**(- 99)
        END
                ERROR CODE=  4
            ***** SEE BASIC TEXTBOOK =  A 322.12  *****
            ***** I WAS AT LINE NUMBER    0
```

sofern die Überschreitung vom FOSBIC-Compiler erfasst wird. Andernfalls geschieht ein maschinenabhängiger Programmabbruch mit dem Hinweis, dass ein „Underflow" bzw. „Overflow" im Zahlenbereich stattfand.

322.2 Alphanumerische Konstante

Alphanumerische Konstante werden auch als Strings bezeichnet. Sie bestehen aus einem Stringklammerpaar und dem von ihm umgrenzten

Stringinhalt. Als Stringklammern sind die BASIC-Zeichen ' und " reserviert.
Die String-Bildungsregel kann generell wie folgt dargestellt werden:

$$\underset{32}{\langle \text{Alphanumerische Konstante} \rangle}\underset{17}{:=}\underset{33}{\langle \text{String-klammer} \rangle}\underset{34}{\langle \text{String-inhalt} \rangle}\underset{33}{\langle \text{String-klammer} \rangle}\,|\,\underset{34}{\langle \text{String-inhalt} \rangle}$$

$$\underset{33}{\langle \text{Stringklammer} \rangle}\underset{32}{:=}\ '\,|\,''$$

$$\underset{34}{\langle \text{Stringinhalt} \rangle}\underset{32}{:=}\underset{1,33}{\left\langle \begin{array}{l}\text{BASIC-}\\ \text{Grundsymbol,}\\ \text{ohne String-}\\ \text{klammer}\end{array}\right\rangle}\,|\,\underset{34}{\langle \text{String-inhalt} \rangle}\underset{1,33}{\left\langle \begin{array}{l}\text{BASIC-}\\ \text{Grundsymbol,}\\ \text{ohne}\\ \text{String-}\\ \text{klammer}\end{array}\right\rangle}$$

Der Stringinhalt (string-constant) muss mindestens aus einem zulässigen BASIC-Grundsymbol bestehen. Falls eine Leerstelle mit dem Symbol ≡ dargestellt wird, weist der einfachste String die Struktur '≡' auf. Derartige Strings dienen – wie noch zu zeigen sein wird[23] – vor allem zur Steuerung der Druckausgabe.
Die Länge einer alphanumerischen Konstanten kann maximal bis zur Vollausschöpfung der für ein BASIC-Statement verfügbaren Zeichenzahl ausgedehnt werden.[24]
Beim FOSBIC-Compiler kann pro BASIC-Statement eine 80-Spaltenlochkarte voll benutzt werden, so dass mittels

DATA ⟨alphanumerische Konstante⟩

der Stringinhalt maximal 74 Zeichen (80 minus Zeichen für DATA und Stringklammern) umfassen kann. Bei Einfügung von Statementnummer und Zwischenräumen – Leerzeichen ausserhalb der alphanumerischen Konstanten – wird die zur Stringdarstellung benutzbare Zeichenzahl entsprechend reduziert[25].
Vermerkt sei, dass die Strings bei der Auflistung der Statements zur Programmkontrolle mit den Stringklammern gedruckt werden; bei der Programmbearbeitung erscheint dagegen nur der Stringinhalt im Ausdruck.

23 Vgl. Abs. 332.21.
24 ANSI Minimal BASIC, p. 12: "The length of a string-constant, that is, the number of characters contained between the quotation-marks, is limited only by the length of a line."
25 Vgl. auch Garland, Dartmouth BASIC, p. 17: "The limit of the length of a statement line implicitly imposes a limit on the length of a string constant."

Teilweise werden die Stringklammern maschinenspezifisch durch andere Zeichen, bei der CD 3300 beispielsweise durch #, dargestellt. Darauf ist beim Studium der in dieser Arbeit abgedruckten BASIC-Programme besonders zu achten.
So wird beispielsweise das Statement

200 PRINT'CD 3300 STRINGDARSTELLUNG'

in der Form

```
200  PRINT #CD 3300 STRINGDARSTELLUNG#
```

gelistet und als

```
CD 3300 STRINGDARSTELLUNG
```

ausgedruckt.
Einzelne BASIC-Compiler lassen in Verbindung mit dem INPUT-Statement auch die Dateneingabe von alphanumerischen Konstanten ohne Stringklammern via data-Statement zu[26].
Vom FOSBIC-Compiler wird die Auslassung einer Stringklammer als Programmierfehler erkannt und durch

```
        THE QUOTATION MARKS ABOVE ARE NOT PAIRED
SEE BASIC TEXTBOOK  A  322.2
```

angezeigt. Fehlen beide Stringklammern, so vermag der FOSBIC-Compiler den Statementinhalt nicht im Sinne des Programmierers als alphanumerische Konstante zu erkennen; die Zeichenkette wird vielmehr als nicht FOSBIC-konform strukturierter Ausdruck interpretiert und die Fehlermeldung

```
        THERE IS SOMETHING I DO NOT LIKE ABOUT THE EXPRESSION SHOWN BELOW
SEE BASIC TEXTBOOK  A  324.1
```

ausgedruckt.

323 Variable

BASIC-Variable werden als Speicherbereiche (Adressen) definiert, denen während des Programmablaufs numerische oder alphanumerische Inhalte zugeordnet werden können. Dementsprechend ist zwischen numerischen und alphanumerischen Variablen zu unterscheiden, die wiederum als einfache oder indizierte Grössen auftreten können. Vgl. Abbil-

26 Vgl. Dartmouth BASIC-6, im besonderen: Waite and Mather, BASIC, p. 60–61. – Vgl. Abs. 332.12.

dung 15. In der anglo-amerikanischen Literatur sind die Bezeichnungen „numeric-variable" und „string-variable" geläufig.[27]

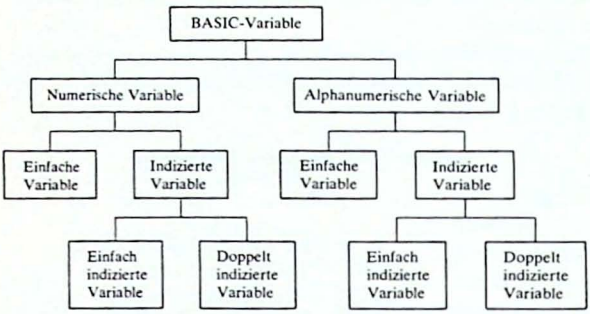


Abbildung 15.

Als metalinguistische Grundgleichung ergibt sich mithin:

$$\underset{35}{\langle \text{Variable} \rangle} := \underset{36}{\left\langle \begin{matrix}\text{Numerische}\\ \text{Variable}\end{matrix} \right\rangle} \mid \underset{44}{\left\langle \begin{matrix}\text{Alphanumerische}\\ \text{Variable}\end{matrix} \right\rangle}$$

Von den bekannten BASIC-Compilern ist nur der UWBIC-Compiler auf die Verarbeitung numerischer Variabler beschränkt.

323.1 Numerische Variable

Numerische Variable können in einfacher oder indizierter Form auftreten, so dass

$$\underset{36}{\left\langle \begin{matrix}\text{Numerische}\\ \text{Variable}\end{matrix} \right\rangle} := \underset{35}{\left\langle \begin{matrix}\text{Einfache numerische}\\ \text{Variable}\end{matrix} \right\rangle} \underset{37|38}{\mid} \underset{39}{\left\langle \begin{matrix}\text{Indizierte numerische}\\ \text{Variable}\end{matrix} \right\rangle}$$

zutrifft.

27 "Variables are associated with either numeric or string values and, in the case of numeric variables, can be either simple variables, which refer to single values, or references to elements of one- or two-dimensional arrays; such references are called subscripted variables." ANSI Minimal BASIC, p. 13. – Vgl. auch Garland, Dartmouth BASIC, p. 13.

Zu beachten ist, dass im Rahmen der Programmiersprache BASIC keine weitere Spezifikation der numerischen Variablen (Integer-/Realgrössen) erfolgt, woraus sich – etwa im Vergleich zu FORTRAN und ALGOL – programmtechnisch wesentliche Vereinfachungen und Erleichterungen ergeben.[28]

323.11 Einfache numerische Variable

Einfache numerische Variable sind normaliter – auch bei ANSI Minimal BASIC[29] – nach der Regel

$$\underset{37}{\left\langle \begin{matrix}\text{Einfache numerische}\\ \text{Variable}\end{matrix}\right\rangle} := \underset{36}{\left\langle \begin{matrix}\text{Gross-}\\ \text{buchstabe}\end{matrix}\right\rangle} \;\Big|\; \underset{3a}{\left\langle \begin{matrix}\text{Gross-}\\ \text{buchstabe}\end{matrix}\right\rangle} \underset{5}{\langle \text{Ziffer} \rangle}$$

zu benennen, während für BASIC-Compiler mit ausgeweitetem Alphabet in Ausnahmefällen

$$\underset{38}{\left\langle \begin{matrix}\text{Einfache numerische}\\ \text{Variable}\end{matrix}\right\rangle} := \underset{36}{\langle \text{Buchstabe} \rangle} \;\Big|\; \underset{3}{\langle \text{Buchstabe} \rangle} \underset{5}{\langle \text{Ziffer} \rangle}$$

gilt, so dass die Maximalzahl der BASIC-konformen Variablennamen von 286 (A, A0, A1, ..., B, B0, ..., Z8, Z9) auf 572 (= Zahl der Buchstaben* 11) ansteigt.
Die einfachen numerischen Variablen werden bei allen BASIC-Compilern implizit – bei erstmaligem Erscheinen im Quellenprogramm – deklariert und bei der Programmkompilation in der Regel auf Null gesetzt.[30] Explizite Nullsetzung ist generell – insbesondere bei Austauschprogrammen – empfehlenswert;[31] sie ist durch Initialisierung der Variablen mittels

LET ⟨Einfache numerische Variable⟩ = 0

vorzunehmen.

28 "There is no distinction between real and integer valued numeric variables as in FORTRAN or ALGOL; hence there is no confusion that 3.0/6 = .5 while 3/6 = 0.0." Garland, Dartmouth BASIC, p. 13.

29 ANSI Minimal BASIC, p. 13.

30 Nach ANSI Minimal BASIC (p. 13) wird empfohlen, alle Variablen erkennbar undefiniert zu lassen. "It is recommended that... all variables are recognizably undefined in the sense that an error will result from any attempt to access the value of a variable before the variable is explicitly assigned a value."

31 ANSI Minimal BASIC, p. 13: "Since initialization of variables is not specified, and hence may vary from implementation to implementation, programs that are intended to be transportable should explicitly assign a value to each variable before any expression involving that variable is evaluated."

Der FOSBIC-Compiler lässt nur numerische Variable nach der Definitionsgleichung 37 zu. Wird die entsprechende Bildungsregel nicht eingehalten, so erfolgt eine Fehlermeldung.
Beispielsweise ist das Statement

```
LET P1 = 3.14
```

FOSBIC-konform; mit ihm wird der Pl bezeichneten numerischen Variablen der Wert 3.14 zugewiesen und gleichzeitig der frühere Speicherinhalt gelöscht. Im weiteren Programmablauf kann über die Variable Pl stets auf den Wert 3.14 zurückgegriffen oder aber Pl durch entsprechende Anweisung zur Speicherung anderer numerischer Daten verwendet werden.
Das nicht BASIC-konforme Statement

```
LET PI = 3.14
```

würde vom FOSBIC-Compiler mit dem Hinweis

```
        ITEM TO THE LEFT OF THE EQUAL SIGN MUST BE A VARIABLE
SEE BASIC TEXTBOOK  A  332.3
```

versehen und das gesamte BASIC-Quellenprogramm – falls anderweitig korrekt – mit dem abschliessenden Satz

```
****** SORRY BUT THERE ARE    1 ERRORS I CANNOT OVERCOME
       BETTER LUCK NEXT TIME
```

als noch nicht lauffähig erklärt.

323.12 Indizierte numerische Variable

Indizierte numerische Variable (subscripted variables) setzen sich aus Feldnamen und Indexliste zusammen, so dass

$$\underset{39}{\langle \text{Indizierte numerische Variable} \rangle} \underset{36}{:=} \underset{40|41|41a}{\langle \text{Feldname} \rangle} \underset{42}{\langle \text{Indexliste} \rangle}$$

mit, nach Minimal BASIC, 31a)

$$\underset{40}{\langle \text{Feldname} \rangle} \underset{39}{:=} \underset{3a}{\langle \text{Grossbuchstabe} \rangle}$$

oder, nach ANSI BASIC, 31b)

31a ANSI Minimal BASIC, p. 13.
31b ANSI BASIC, p. 18.

$$\underset{41}{\langle \text{Feldname} \rangle} \underset{39}{:=} \underset{3a}{\left\langle \begin{matrix}\text{Gross-}\\ \text{buchstabe}\end{matrix} \right\rangle} \mid \underset{3a}{\left\langle \begin{matrix}\text{Gross-}\\ \text{buchstabe}\end{matrix} \right\rangle} \underset{5}{\langle \text{Ziffer} \rangle}$$

gilt. Für BASIC-Prozessoren mit ausgeweitetem Alphabet trifft

$$\underset{41a}{\langle \text{Feldname} \rangle} \underset{39}{:=} \underset{2}{\langle \text{Alphazeichen} \rangle}$$

zu. Weiterhin bestehen die Definitionsgleichungen

$$\underset{42}{\langle \text{Indexliste} \rangle} \underset{39}{:=} (\underset{43}{\langle \text{Index} \rangle}) \mid (\underset{43}{\langle \text{Index} \rangle}, \underset{43}{\langle \text{Index} \rangle}) \mid$$

$$(\underset{43}{\langle \text{Index} \rangle}, \underset{43}{\langle \text{Index} \rangle}, \underset{43}{\langle \text{Index} \rangle})$$

und

$$\underset{43\ 42}{\langle \text{Index} \rangle} := \underset{49}{\left\langle \begin{matrix}\text{Arith-}\\ \text{metischer}\\ \text{Ausdruck}\end{matrix} \right\rangle}$$

Nach ANSI Minimal BASIC sind Feldnamen nur durch Buchstaben zu bilden; dies gilt auch für den FOSBIC-Compiler. Dabei ist zu beachten, dass derselbe Buchstabe nicht gleichzeitig zur Bezeichnung einer einfachen und einer indizierten numerischen Variablen verwendet werden darf. Die Anzahl der einfachen und indizierten numerischen Variablen ist damit insgesamt auf maximal 26 bzw. auf die Anzahl der Alphazeichen begrenzt.[32]

Der Index einer indizierten Variablen kann selbst als indizierte Variable oder als arithmetischer Ausdruck definiert werden; die sich dabei ergebenden effektiven Indexwerte müssen ganzzahlig und nicht negativ sein, wobei Gleitkommazahlen compilerspezifisch entweder zur nächsten Ganzzahl gerundet oder durch Abbruch des Dezimalteils ganzzahlig gemacht werden. Nach ANSI Minimal BASIC und ANSI BASIC darf die Indexliste nur zwei Indizes enthalten[32a]; BASIC-Compiler auf Grossrechnersystemen erlauben jedoch teilweise drei Indizes.[32b]

Das einfach und doppelt indizierte Variable umfassende BASIC-Statement

32 ANSI Minimal BASIC, p. 13.
32a ANSI Minimal BASIC, p. 13; ANSI BASIC, p. 18.
32b Control Data, BASIC Version 3, p. 3.2.

```
70 LET A(I-7/N) = B(3) + C(R(K + L(2*M)), (I + 3))
```

einfach

indizierte Variable

doppelt

ist syntaktisch korrekt und bewirkt Zuweisung des Additionsergebnisses an den Speicherplatz A(I–7/N). Dabei ist zu beachten, dass der Ausdruck (I–7/N) auch einen nicht ganzzahligen Wert ergeben kann, der sich – falls nicht negativ – durch Rundung/Abbruch des Dezimalteils in einen zulässigen Indexwert umsetzen lässt. So würde etwa (I–7/N) = 3.14 jedenfalls den Indexwert 3 und (1–7/N) = 0.98 den Indexwert 1 resp. 0 ergeben. Nach ANSI Minimal BASIC wird der Indexwert unter Rundung des Dezimalteils bestimmt, wobei der gerundete Wert von X als INT(X+0.5) definiert ist.[33]

Beim FOSBIC-Compiler wird die Umsetzung von Gleitkommazahlen in Ganzzahlen durch Abbruch des Dezimalteils (truncation) vorgenommen. Ohne begleitendes DIM-Statement[34] werden die indizierten Variablen – mit erstmaligem Erscheinen im Programm – implizit dimensioniert. Nach ANSI Minimal BASIC kann diesfalls der Index von 0 bis einschliesslich 10 laufen, so dass für einfach indizierte numerische Variable 11 und für doppelt indizierte numerische Variable 121 Speicherplätze reserviert werden.[35]

Vom FOSBIC-Compiler werden bei implizierter Dimensionierung für einfach und doppelt indizierte Variable generell 121 Speicherplätze reserviert. Ihre Zahl kann durch explizite Dimensionierung – Einfügung von DIM-Statements in das Quellenprogramm – den problemspezifischen Anforderungen entsprechend erhöht oder, insbesondere zur Speicherplatzeinsparung, gesenkt werden.

Vom FOSBIC-Compiler werden bei der Programmumwandlung[36] Feldnamen, Indexlisten und Indizes auf syntaktische Fehler überprüft und gegebenenfalls Fehlermeldungen vom Typ A, G oder I sowie J, K oder L protokolliert.

```
A -- THIS IS AN ILLEGAL NAME

G -- A CONSTANT CANNOT BE FOLLOWED BY A LEFT PARENTHESIS

I -- A RIGHT PARENTHESIS CANNOT BE FOLLOWED BY A VARIABLE OR CONSTANT

J -- THESE VARIABLES ARE NEXT TO EACH OTHER

K -- THESE CONSTANTS ARE NEXT TO EACH OTHER

L -- A VARIABLE IS NEXT TO A CONSTANT
```

33 Vgl. Abs. 325.1.
34 Vgl. Abs. 331.21.
35 ANSI Minimal BASIC, p. 23.
36 Vgl. Abs. 233.

Bei nicht spezifizierbaren syntaktischen Fehlern erscheint die Meldung F,

```
F -- THERE IS SOMETHING FUNNY HERE
```

Das folgende Beispiel mag die Struktur dieser Fehlermeldungen aufzeigen.

```
        70 LET A1(I-7)/N = BI(3) + (R+L(2*M), I+0(
        I DO NOT LIKE THE EXPRESSION SHOWN BELOW (REASONS FOLLOW)
SEE BASIC TEXTBOOK  A  324.1

            BI(3)+(R+L(2*M),I+0(
            AA     B        D   B
                 A -- THIS IS AN ILLEGAL NAME

                 B -- NO RIGHT PARENTHESIS FOR THIS LEFT PARENTHESIS

                 D -- THIS COMMA IS IN A FUNNY PLACE

        ITEM TO THE LEFT OF THE EQUAL SIGN MUST BE A VARIABLE
SEE BASIC TEXTBOOK  A  332.3
```

323.2 Alphanumerische Variable

Analog zu den numerischen Variablen können – vgl. Abbildung 15 – auch die alphanumerischen Variablen (string-variables) in zwei Kategorien eingeteilt werden.

$$\underset{44}{\left\langle \begin{array}{l}\text{Alphanumerische}\\ \text{Variable}\end{array}\right\rangle} \underset{35}{:=} \underset{45}{\left\langle \begin{array}{l}\text{Einfache alpha-}\\ \text{numerische}\\ \text{Variable}\end{array}\right\rangle} \Bigg| \underset{46}{\left\langle \begin{array}{l}\text{Indizierte alpha-}\\ \text{numerische}\\ \text{Variable}\end{array}\right\rangle}$$

323.21 Einfache alphanumerische Variable

Einfache alphanumerische Variable werden normaliter nach der Regel[37]

$$\underset{45}{\left\langle \begin{array}{l}\text{Einfache alpha-}\\ \text{numerische}\\ \text{Variable}\end{array}\right\rangle} \underset{44}{:=} \underset{37|38}{\left\langle \begin{array}{l}\text{Einfache}\\ \text{numerische}\\ \text{Variable}\end{array}\right\rangle} \$ \Bigg| \underset{37|38}{\left\langle \begin{array}{l}\text{Einfache}\\ \text{numerische}\\ \text{Variable}\end{array}\right\rangle} \$ \underset{47}{\left\langle \begin{array}{l}\text{Teilzeichen-}\\ \text{ketten-}\\ \text{begrenzung}\end{array}\right\rangle}$$

gebildet, wobei die Anwendung des ausgeweiteten Alphabets wiederum nur compilerspezifisch zugelassen ist. Die Teilzeichenkettenbegrenzung

37 Nach ANSI Minimal BASIC, p. 13 gilt nur: "String-variables consist of a letter followed by a dollar sign."

ermöglicht es, Teile des in einer alphanumerischen Variablen enthaltenen Strings innerhalb einer unteren und oberen Grenze anzusprechen.
Offensichtlich unterscheiden sich die einfachen alphanumerischen Variablen strukturell nur durch das beigefügte Dollarzeichen ($) von den einfachen numerischen Variablen. Beide Variablentypen sind vollständig voneinander unabhängig; insbesondere bestehen auch zwischen numerischen und alphanumerischen Variablen gleichen Namens keine Beziehungen und damit auch keinerlei Beschränkungen hinsichtlich ihrer Parallelverwendung.[38]
Nach ANSI Minimal BASIC soll der für eine alphanumerische Variable reservierte Speicherplatz bis zu 18 Zeichen aufzunehmen vermögen, wobei die effektive Zeichenzahl während des Programmablaufs variieren kann. Einzelne Compiler lassen bedeutend grössere Stringlängen zu; so kann bei Dartmouth BASIC-6 ein String bis zu 4095 Zeichen umfassen.[39]
Die Wertzuweisung erfolgt durch entsprechende LET-, READ- oder INPUT-Statements,[40] zum Beispiel mit

```
10 LET B$=≠START≠
```

eim FOSBIC-Compiler ist der einer einfachen alphanumerischen Variablen zuweisbare Stringinhalt auf 5 Zeichen beschränkt, wobei – mit Ausnahme der Stringklammern – alle BASIC-Grundsymbole zur Bildung des Stringinhalts verwendbar sind.
Bezüglich des FOSBIC-Compilers ist zu beachten, dass der Programmablauf nicht unterbrochen wird, falls einer numerischen Variablen ein alphanumerischer Wert zugewiesen wird; diesfalls wird – wie das folgende Beispiel zeigt – die Eingabeanweisung lediglich warnend vermerkt.

```
20 LET C=≠ENDE≠
YOU TREAT A NUMERIC VARIABLE AS AN ALPHANUMERIC ONE
SEE BASIC TEXTBOOK  A  323.1
```

Intern bestehen beim FOSBIC-Compiler keine Unterschiede zwischen numerischen und alphanumerischen Variablen. Eine Doppelverwendung

38 ANSI Minimal BASIC, p. 13: "There is no relationship between a numeric-variable and a string-variable whose representations agree except for the dollar sign."
39 Waite and Mather, BASIC, p. 162. – Vgl. auch Garland, Dartmouth BASIC, p. 15: "Strings may have arbitrary length up to 4095 characters. Earlier versions of BASIC imposed more severe limitations on string length, e.g. 15 characters, in order to simplify the problem of storage management. In any implementation, strings should be of variable rather than fixed length."
40 Vgl. Abs. 332.31, 332.13 und 332.14.

ist zulässig, muss aber in der Logik des Programmablaufs berücksichtigt werden. Die Bildung von Teilzeichenketten ist nicht möglich, ebensowenig eine Veränderung der speicherbaren Stringlänge je Variable.
Die FOSBIC-Version 2 ist vollständig den zukünftigen Textverarbeitungsmöglichkeiten angepasst, indem zunächst eine eindeutige Unterscheidung zwischen numerischen und alphanumerischen Variablen getroffen wird. Mittels expliziter Dimensionierung kann die speicherbare Zeichenkette beliebig über den impliziten Wert von 9 Zeichen hinaus vergrössert werden. Lediglich eventuelle physikalische Grenzen der verfügbaren peripheren Speicher limitieren diese Angaben. Die Bildung von Teilzeichenketten ist möglich.

323.22 Indizierte alphanumerische Variable

Die Bildungsregel für indizierte alphanumerische Variable kann generell wie folgt dargestellt werden:

$$\underset{46}{\langle \text{Indizierte alphanumerische Variable} \rangle_{44}} := \underset{40|41}{\langle \text{Feldname} \rangle} \$ \underset{42}{\langle \text{Indexliste} \rangle} \mid$$

$$\underset{40|41}{\langle \text{Feldname} \rangle} \$ \underset{47}{\left\langle \begin{matrix} \text{Teilzeichen-} \\ \text{ketten-} \\ \text{begrenzung} \end{matrix} \right\rangle} \underset{42}{\langle \text{Indexliste} \rangle} \mid$$

$$\underset{40|41}{\langle \text{Feldname} \rangle} \$ \underset{42}{\langle \text{Indexliste} \rangle} \underset{47}{\left\langle \begin{matrix} \text{Teilzeichen-} \\ \text{ketten-} \\ \text{begrenzung} \end{matrix} \right\rangle}$$

Der alphanumerische Charakter einer indizierten Variablen wird mithin wiederum durch das dem Feldnamen folgende Dollarzeichen ausgewiesen.
Die Verarbeitung indizierter alphanumerischer Variabler ist nach ANSI Minimal BASIC nicht vorgesehen, bei manchen BASIC-Compilern aber möglich; vgl. Tabelle 4. Der FOSBIC-Compiler ist auf feste Stringlängen und das Fehlen der Teilzeichenkettenbegrenzung ausgelegt. Mit der Version 2 sind diese Restriktionen weggefallen.

Von ANSI BASIC wird auch die Verarbeitung indizierter alphanumerischer Variabler geregelt.[40a]

323.23 Teilzeichenbegrenzung alphanumerischer Variabler

Um Teilzeichenketten aus der gespeicherten Gesamtzeichenkette einer einfachen oder indizierten alphanumerischen Variablen herauszugreifen, wird nach ANSI BASIC eine sogenannte Teilzeichenbegrenzung (subsring-qualifier) vorgeschlagen.[40b]
Der syntaktische Aufbau der Teilzeichenkettenbegrenzung erfolgt nach der Regel

$$\underset{47}{\left\langle \begin{matrix}\text{Teilzeichen-}\\ \text{ketten-}\\ \text{begrenzung}\end{matrix}\right\rangle} \underset{45|46}{:=} \left(\left\langle \begin{matrix}\text{Untere}\\ \text{Begren-}\\ \text{zung}\end{matrix}\right\rangle_{47a} \left\langle \begin{matrix}\text{Teilungs-}\\ \text{zeichen}\end{matrix}\right\rangle_{47c} \left\langle \begin{matrix}\text{Obere}\\ \text{Begren-}\\ \text{zung}\end{matrix}\right\rangle_{47b}\right)$$

$$\underset{47a}{\left\langle \begin{matrix}\text{Untere}\\ \text{Begrenzung}\end{matrix}\right\rangle} \underset{47}{:=} \underset{49}{\left\langle \begin{matrix}\text{Arithmetischer}\\ \text{Ausdruck}\end{matrix}\right\rangle}$$

$$\underset{47b}{\left\langle \begin{matrix}\text{Obere}\\ \text{Begrenzung}\end{matrix}\right\rangle} \underset{47}{:=} \underset{49}{\left\langle \begin{matrix}\text{Arithmetischer}\\ \text{Ausdruck}\end{matrix}\right\rangle}$$

$$\underset{47c}{\left\langle \begin{matrix}\text{Teilungs-}\\ \text{zeichen}\end{matrix}\right\rangle} \underset{47}{:=} \quad :\,|\,,$$

Nach ANSI BASIC ist als Teilungszeichen nur der Doppelpunkt (:) vorgesehen. BASIC-Prozessoren, die keine indizierten alphanumerischen Variablen zulassen, verwenden jedoch auch das Komma (,) als Teilungszeichen, was leider zu Verwechslungen mit einer Indexliste führen

40a ANSI BASIC, p. 35: "Subscripted string-variables are named by a letter followed by an optional digit followed by a dollar-sign followed by one or two numeric-expressions enclosed within parentheses. Explicit declarations of string-variables are not required: a dollar-sign serves to distinguish a string-variable from a numeric-variable, and the presence of a subscript distinguishes a subscripted variable from a simple one."
40b ANSI BASIC, p. 37.

kann.[40c] Die Teilzeichenkettenbegrenzung wird im allgemeinen an den alphanumerischen Feldnamen – direkt hinter das Dollarzeichen – angehängt.
Bei der FOSBIC-Version 2 kann die Teilzeichenkettenbegrenzung auch an die Indexliste angefügt werden.
Wenn zum Beispiel die einfache alphanumerische Variable A$ den Textinhalt

A$ = "ABCDE"

hat, so erzeugen die nachstehenden Teilzeichenkettenbegrenzungen folgende "substrings":

A$(1:1) = "A"
A$(1:2) = "AB"
A$(0:3) = "ABC"
A$(3:6) = "DE"
A$(3:1) = ""
A$(6:9) = ""
A$(9:3) = ""

Dabei bezeichnet "" einen Leerstring der Länge 0.
Aus diesen Beispielen folgt, dass bei einer Unterschreitung der unteren Teilzeichenbegrenzung stets die Untergrenze und bei einer Überschreitung stets die Obergrenze angenommen wird. Sich widersprechende Kombinationen der Begrenzungsbereiche – bei denen die untere Begrenzung grösser als die obere ist oder der Begrenzungsbereich ausserhalb der maximalen Stringlänge liegt – führen zur Übertragung eines Stringendezeichens, so dass ein String der Länge 0 entsteht.[40d]
Die FOSBIC-Version 2 realisiert den von ANSI BASIC angestrebten Standard vollumfänglich.

40c Zum Beispiel Nixdorf, System-Software Business-BASIC, S. 4.18–4.22.

40d ANSI BASIC, p. 37–38: "The substring-qualifier provides a means for specifying a portion of the value associated with a string-variable. A$(M:N) shall specify that substring of the value associated with A$ from its Mth to its Nth characters inclusive (M and N are indices). Characters in a string shall be numbered from the left starting with one. There are no exceptions associated with substring-qualifiers: if either M or N is not in the range from 1 to LEN(A$), then M shall be considered to be MAX(N,1) and N shall be considered to be MIN(N,LEN(A$)). If M < N, even after this adjustment, then A$(M:N) shall be the null string. E.g., if A$ = "1234", then A$(1:1) = "1", A$(1:3) = "123", A$(0:3) = "123", A$(2:5) = "234", and A$ (3:2) = A$(5:7) = ""."

Grundsätzlich ist bei der Programmiersprache BASIC zwischen arithmetischen, alphanumerischen und logisch-komparativen Ausdrücken zu unterscheiden; vgl. Abbildung 16. Die metalinguistische Grundgleichung für BASIC-Ausdrücke lautet mithin:

$$\underset{48}{\langle}\text{Ausdruck}\underset{75}{\rangle} := \langle\text{Arithmetischer Ausdruck}\rangle_{49} \mid \langle\text{Alphanumerischer Ausdruck}\rangle_{50|51} \mid \langle\text{Logisch-komparativer Ausdruck}\rangle_{52}$$

Tabelle 4. Variablenformen und Variablengrenzwerte

Prozessor	Numerische Variable	Alphanumerische Variable			
		einfache Variable Form[1]	Zeichen max.	einfach indizierte Variable Zeichen max.	doppelt indizierte Variable Zeichen max.
ANSI Minimal BASIC	Für numerische Variable gelten dieselben Grenzwerte wie für numerische Konstante, siehe Tabelle 3; verfügbar sind bei allen Compilern alle angegebenen Formen numerischer Variabler.	⟨B⟩$[2]	18	unzulässig	unzulässig
Dartmouth BASIC-6		45	Bei der Compilierung bis 120; während der Programmausführung bis 4095.		
UOM BASIC/ UBASIC		45	511	511	unzulässig
IBM CALL BASIC		46	18	18	unzulässig
UWBIC		unzulässig		unzulässig	unzulässig
FOSBIC		45	5	5	5

[1] Die Zahlenangaben für die Form beziehen sich auf die Nummer der Definitionsgleichung.
[2] B steht hier für Buchstabe!

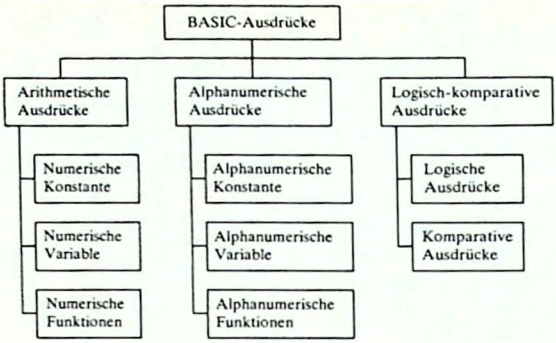


Abbildung 16.

324.1 Arithmetische Ausdrücke

Arithmetische Ausdrücke sind alleinstehende oder durch dyadische Operatoren miteinander verbundene numerische Konstante, Variable und/oder Funktionen.[41]

⟨Arithmetischer Ausdruck⟩ (49, 48) : = ⟨Numerische Konstante⟩ (18) | ⟨Numerische Variable⟩ (36) | ⟨Numerische Funktion⟩ (56) |

⟨Numerische Hybrid-funktion⟩ (68b) | (⟨Arithmetischer Ausdruck⟩ (49)) |

⟨Monadischer Operator⟩ (11) ⟨Arithmetischer Ausdruck⟩ (49) |

⟨Arithmetischer Ausdruck⟩ (49) ⟨Dyadischer Operator⟩ (12) ⟨Arithmetischer Ausdruck⟩ (49)

Arithmetische Ausdrücke treten vor allem in Wertzuweisungs-/Berechnungsanweisungen auf, beispielsweise in der Form
LET ⟨Numerische Variable⟩ = ⟨Arithmetischer Ausdruck⟩.[42]

41 ANSI Minimal BASIC, p. 14–15. Garland, Dartmouth BASIC, p. 18–20.
42 Vgl. Abs. 332.311.

Durch dyadische Operatoren miteinander verknüpfte arithmetische Ausdrücke werden nach der Prioritätsregel

Exponentiation vor Multiplikation und Division vor Addition und Subtraktion

abgearbeitet, sofern diese Regel nicht durch Klammersetzung aufgehoben wird.
Operatoren der gleichen Prioritätsstufe werden – aus der Sicht des BASIC-Benutzers[43] – von links nach rechts abgearbeitet, so dass etwa A−B−C als (A−B)−C, A/B/C als (A/B)/C und A**B**C als (A**B)**C zu interpretieren sind.
Klammern werden generell von innen nach aussen und Funktionen[44] von rechts nach links verarbeitet.
Die vorerwähnten Prioritätsregeln werden in Abbildung 17 schematisch dargestellt und anhand der Formel

$\sqrt{3 - B \cdot 8 - \frac{A^3}{2}}$, in BASIC-Schreibweise SQR(3−(B*8)−A**3/2),

exemplifiziert.
Die Eingliederung monadischer Operatoren ist normalerweise aufgrund der mathematischen Symbolik eindeutig. Zu beachten ist allerdings, dass monadische Operatoren bei BASIC-konformer Darstellung nur das am weitesten links stehende oder aber das einer linken Klammer unmittelbar folgende Zeichen eines arithmetischen Ausdrucks sein dürfen. Mithin sind die Ausdrücke −A*(−B) und A**(−2) BASIC-konform, nicht dagegen die entsprechenden klammerfreien Darstellungen.
Dagegen sollte der Ausdruck −A**2 von allen BASIC-Compilern richtig verarbeitet werden. Dies ist – worauf John Sack und Judith Meadows hinweisen[45] – nicht immer der Fall, indem das Testprogramm

```
10  LET A = -4
20  LET B = -A**2
30  PRINT B
40  END
```

43 ANSI Minimal BASIC, p. 14.
44 Vgl. Abs. 325.
45 John R. Sack and Judith L. Meadows. Entering BASIC. Chicago 1973, p. 35. – Ähnlich auch: A. I. Forsythe, C. E. Hughes, R. M. Aiken and E. I. Organick. Computer Science. Programming in BASIC. New York 1976, p. 37–38.

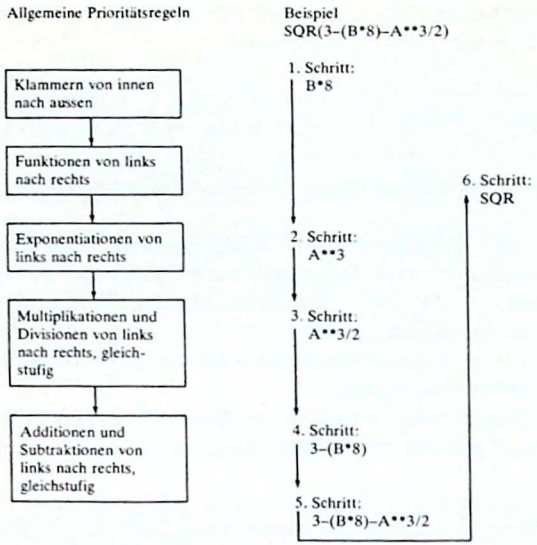


Abbildung 17. Bearbeitung von BASIC-Ausdrücken

bei einzelnen Compilern nicht −16, sondern compilerspezifisch +16 ergibt. Darauf macht auch Terry Walker aufmerksam und bemerkt, daß ein eventueller Compilerfehler vom BASIC-Benutzer durch freizügigen Klammergebrauch überbrückbar sei.[46] Im vorstehenden Beispiel wäre das zweite LET-Statement durch

```
20  LET B=(-A)**2
```

zu ersetzen. Der FOSBIC-Compiler verarbeitet die monadischen Operatoren jedenfalls korrekt, wobei die Arithmetik mit der von ALGOL und FORTRAN übereinstimmt. Zusätzlich kann der FOSBIC-Compiler auf die gewünschte Verarbeitungsart der monadischen Operatoren bei der Implementierung eingestellt werden.[47]

46 An das erwähnte Zahlenbeispiel angepasst, lautet der entsprechende Hinweis von T. Walker wie folgt: "The evaluation of unary minus operators varies among different dialects. For example, in some dialects the expression −4**2 would produce a value of 16, while in other dialects the result would be −16. The difference in answers is because, in the first case the expression is evaluated as (−4)**2, while in the second case it is evaluated as −(4**2). To avoid this ambiguity, parentheses should be used to show the desired order of evaluation." Terry M. Walker. Introduction to Computer Science: BASIC Language Programming. Boston 1973, p. 45.

47 Vgl. Bd. 2, Abs. 532.1.

Freizügiger Klammergebrauch ist nicht nur im vorerwähnten – generell beachtenswerten – Fall am Platz. Vielmehr sollte bei Übernahme der in mathematischer Schreibweise vorliegenden arithmetischen Ausdrücke in BASIC-Programmen immer dafür gesorgt werden, daß Unklarheiten bezüglich der Abarbeitung komplexer Ausdrücke ausgeschaltet werden. Einige Beispiele für BASIC-konforme Umschreibung mathematischer Ausdrücke seien hier angeführt.

Mathematische Schreibweise	BASIC-Schreibweise
$3/-4$	3/(–4)
$-ba$	–B*A
$a(-b)$	A*(–B)
$\frac{a+b}{c-d}$	(A+B)/(C–D)
$(ab)-(cd)$	(A*B)–(C*D)
$\frac{-b+\sqrt{b^2-4ac}}{2a}$	(–B+((B**2–4*A*C)**(1/2)))/(2*A) oder (–B+SQR(B**2–4.0*A*C))/(2.0*A)
$a^{1/2}$	A**(1/2) oder A**.5 oder SQR(A)

Besonders zu beachten ist, dass das Multiplikationszeichen (*) immer gesetzt werden muss. Ein aus mehreren Konstanten und Variablen bestehender Nenner ist ebenfalls immer in Klammern zu setzen. Beim Potenzieren ist der compilerspezifische Operator (^ oder **) zu wählen.
Bezüglich der Exponentiation sei speziell vermerkt, dass sich im allgemeinen keine BASIC-spezifischen Probleme ergeben, solange der Exponent eine Ganzzahl ist; insbesondere gilt auch 0**0 = 1. Falls der Exponent eine Gleitkommazahl darstellt, kann im Falle eines negativen Arguments als Ergebnis eine imaginäre Zahl entstehen, wie das Beispiel

$-4**(1/2) = \sqrt{-4} = 2\sqrt{-1} = 2i$ zeigt, während
$-8**(1/3) = \sqrt[3]{-8} = -2$ ergibt.

Die Behandlung dieser Fälle erfolgt nicht bei allen BASIC-Compilern gleich. Bei Dartmouth BASIC-6 wird die Meldung „POWER OF NEGATIVE NUMBER" ausgedruckt und das Programm mit dem sich er-

gebenden Absolutwert fortgeführt; beispielsweise würde ABS(−3)^.7 für (−3)^.7 gesetzt.[48]
Der FOSBIC-Compiler prüft bei nichtganzzahligem Exponenten zunächst das Argument auf ein negatives Vorzeichen ab. Gegebenenfalls wird ABS (Argument)**Exponent = Ergebnis und (−Ergebnis)**(1/Exponent) = Prüfwert berechnet und nachher geprüft, ob Prüfwert = Argument ist. Zutreffendenfalls wird das Ergebnis mit einem negativen Vorzeichen versehen und im Programm fortgefahren, andernfalls erfolgt die Fehlermeldung.

```
PROGRAM STOPPED -- ILLEGAL EXPONENTIATION -- THE ARGUMENT IS NEGATIVE
                   0.39560357E+34 THE EXPONENT IS  0.12345000E+05
                   ERROR CODE= 15
              ***** SEE BASIC TEXTBOOK =  A  324.1  *****
              ***** I WAS AT LINE NUMBER   10
```

Bezüglich der FOSBIC-Fehlermeldungen für arithmetrische Ausdrücke gilt generell, dass anschliessend an den Satz

```
          I DO NOT LIKE THE EXPRESSION SHOWN BELOW (REASONS FOLLOW)
SEE BASIC TEXTBOOK  A  324.1
```

der fehlerhafte Teil des Ausdrucks noch einmal gelistet wird, wobei die darunter erscheinenden Buchstaben den jeweiligen Fehlerursachen A bis L zuzuordnen sind.

```
A -- THIS IS AN ILLEGAL NAME

B -- NO RIGHT PARENTHESIS FOR THIS LEFT PARENTHESIS

C -- NO LEFT PARENTHESIS FOR THIS RIGHT PARENTHESIS

D -- THIS COMMA IS IN A FUNNY PLACE

E -- THIS PAIR OF OPERATORS IS ILLEGAL

F -- THERE IS SOMETHING FUNNY HERE

G -- A CONSTANT CANNOT BE FOLLOWED BY A LEFT PARENTHESIS

H -- THIS OPERATOR IS DANGLING

I -- A RIGHT PARENTHESIS CANNOT BE FOLLOWED BY A VARIABLE OR CONSTANT

J -- THESE VARIABLES ARE NEXT TO EACH OTHER

K -- THESE CONSTANTS ARE NEXT TO EACH OTHER

L -- A VARIABLE IS NEXT TO A CONSTANT
```

Kann ein syntaktischer Fehler nicht im vorerwähnten Sinne spezifiziert werden, so erscheint der generelle Hinweis

```
          THERE IS SOMETHING I DO NOT LIKE ABOUT THE EXPRESSION SHOWN BELOW
SEE BASIC TEXTBOOK  A  324.1
```

48 Waite and Mather, BASIC, p. 150.

Falls beispielsweise das mit LET A(I,J) = 3*(B0+SIN(−X)) codierte Statement bei der Programmübertragung durch Tippfehler zu

```
230 LET A(I,J) = 3*(B/+S2N(-X)
```

entstellt wird, erfolgt der Nachweis syntaktischer Fehler in der Form

```
          I DO NOT LIKE THE EXPRESSION SHOWN BELOW (REASONS FOLLOW)
SEE BASIC TEXTBOOK  A  324.1

                3*(B/+S2N(-X)
                  B EEJJJ
                     B -- NO RIGHT PARENTHESIS FOR THIS LEFT PARENTHESIS

                     E -- THIS PAIR OF OPERATORS IS ILLEGAL

                     J -- THESE VARIABLES ARE NEXT TO EACH OTHER
```

324.2 Alphanumerische Ausdrücke

Für alphanumerische Ausdrücke gilt normaliter

$$\left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rischer}\\ \text{Ausdruck}\end{array}\right\rangle_{50} := \left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rische}\\ \text{Konstante}\end{array}\right\rangle_{48}\Bigg|_{32} \left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rische}\\ \text{Variable}\end{array}\right\rangle_{44}\Bigg|$$

$$\left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rische}\\ \text{Funktion}\end{array}\right\rangle_{64}\Bigg| \left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rische}\\ \text{Hybrid-}\\ \text{funktion}\end{array}\right\rangle_{68c}$$

und in Sonderfällen, falls Verkettung zulässig ist,

$$\left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rischer}\\ \text{Ausdruck}\end{array}\right\rangle_{51} := \left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rischer}\\ \text{Ausdruck}\end{array}\right\rangle_{48} \left\langle \begin{array}{l}\text{Alpha-}\\ \text{numerischer}\\ \text{Operator}\end{array}\right\rangle_{51} \left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rischer}\\ \text{Ausdruck}\end{array}\right\rangle_{14} \left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rischer}\\ \text{Ausdruck}\end{array}\right\rangle_{51}$$

Dabei ist als alphanumerischer Operator nach ANSI BASIC nur das Et-Zeichen (&) zulässig.[49]
Ein von der FOSBIC-Version 2 verarbeitetes BASIC-Programm zeigt das nachstehende Beispiel.

49 ANSI BASIC, p. 37.

```
10 READ A$, B$, C$
20 DATA "ING","SHAR","TIME-"
30 LET D$=C$ & B$ & A$
40 PRINT D$
50 END

TIME-SHARING
```

324.3 Logisch-komparative Ausdrücke

Für die Klasse der logisch-komparativen Ausdrücke gilt

$$\underset{52}{\langle \text{Logisch-komparativer Ausdruck} \rangle} := \underset{48}{\langle \text{Logischer Ausdruck} \rangle} \mid \underset{53}{\langle \text{Komparativer Ausdruck} \rangle} \quad \underset{54}{}$$

24.31 Logische Ausdrücke

Logische Ausdrücke können numerische oder alphanumerische Variable betreffen und wie folgt strukturiert sein:

$$\underset{53}{\langle \text{Logischer Ausdruck} \rangle} := \underset{52}{} \underset{54}{\langle \text{Komparativer Ausdruck} \rangle} \underset{15}{\langle \text{Boolscher Operator} \rangle} \underset{54}{\langle \text{Komparativer Ausdruck} \rangle} \mid$$

$$\underset{54}{\langle \text{Komparativer Ausdruck} \rangle} \underset{15}{\langle \text{Boolscher Operator} \rangle} (\underset{53}{\langle \text{Logischer Ausdruck} \rangle}) \mid$$

$$(\underset{53}{\langle \text{Logischer Ausdruck} \rangle}) \underset{15}{\langle \text{Boolscher Operator} \rangle} (\underset{53}{\langle \text{Logischer Ausdruck} \rangle})$$

Logische Ausdrücke werden normalerweise nur im IF-Statement verwendet.[50]
Ist der komparative Ausdruck lediglich ein arithmetischer Ausdruck, so wird diese Kurzform implizit als

$$\langle \text{Arithmetischer Ausdruck} \rangle = 1$$

50 Vgl. Abs. 322.42.

interpretiert. Zutreffendenfalls wird der komparative Ausdruck als wahr (true), sonst als falsch (false) ausgelegt.
Der logische Ausdruck nach Gleichung 53 erweitert sich bei Verwendungsmöglichkeit des impliziten komparativen Ausdrucks zu

$$\underset{53a}{\left\langle \begin{matrix}\text{Logischer}\\ \text{Ausdruck}\end{matrix}\right\rangle} := \underset{53}{\left\langle \begin{matrix}\text{Logischer}\\ \text{Ausdruck}\end{matrix}\right\rangle} \mid \text{NOT} \underset{54}{\left\langle \begin{matrix}\text{Komparativer}\\ \text{Ausdruck}\end{matrix}\right\rangle} \mid$$

$$\text{NOT} \left(\underset{54}{\left\langle \begin{matrix}\text{Komparativer}\\ \text{Ausdruck}\end{matrix}\right\rangle} \right)$$

Beispielsweise ist der Programmausschnitt

```
200  IF NOT X THEN 300
210  IF A AND B THEN 400
220  IF A OR B THEN 500
```

dahingehend zu interpretieren, dass eine Sprunganweisung auszuführen ist, wenn bei Statement 200 „X gleich Null", bei Statement 210 „A und B ungleich Null" als zutreffend erkannt wird.
Einzelne Compiler lassen die Verwendung logischer Ausdrücke auch in Verbindung mit (IF/THEN implizit mitumfassenden) PRINT-Statements zu, wobei durch Druck der Ziffer 1 resp. 0 ein positives resp. negatives Kontrollresultat angezeigt wird.[51] Beispiel:

```
200  PRINT NOT X
210  PRINT A AND B
220  PRINT A OR B
```

Mit den logischen Ausdrücken können unter Verwendung komparativer Ausdrücke komplexe logisch-komparative Zusammenhänge überprüft werden. So erfolgt auf Grund des BASIC-Statements

```
100  IF A>B AND C<B THEN 200
```

der Übergang zum Statement mit der Statementnummer 200 nur dann, wenn die beiden komparativen Ausdrücke zutreffen, d. h. wenn A größer als B und gleichzeitig C kleiner als B ist; andernfalls wird mit dem nächstfolgenden Statement fortgefahren.

51 Hewlett Packard. Time-Sharing BASIC. Nieder-Eschbach 1970, S. E.2–4. – Vgl. auch: Eugene H. Barnett. Programming Time-Shared Computers in BASIC. New York 1972, p. 300–306.

Werden mehrere komparative Ausdrücke mit Boolschen Operatoren zu einer Boolschen Gleichung verknüpft, so empfiehlt sich die Verwendung von Klammern, um eine eindeutige logische Kette zu erhalten. Ohne die Verwendung von Klammern wird eine Boolsche Gleichung von links beginnend nach rechts fortlaufend interpretiert. Zum Beispiel wird die Anweisung

```
20 IF A=1 AND B=1 AND C=1 OR D=4 THEN 259
```

dahingehend verstanden, dass nach der Anweisung mit der Statementnummer 259 dann verzweigt wird, wenn entweder D den Wert 4 enthält, oder wenn gleichzeitig A und B und C den Wert 1 beinhalten. Durch Verwendung von Klammern wird diese logische Bedingung deutlicher.

```
20 IF (A=1 AND B=1 AND C=1) OR D=4 THEN 259
```

Nach ANSI BASIC sind nur logische Ausdrücke nach der Gleichung 53 vorgesehen.

24.32 Komparative Ausdrücke

Komparative Ausdrücke können numerische oder alphanumerische Werte betreffen, so dass

$$\left\langle \begin{array}{l}\text{Komparativer}\\ \text{Ausdruck}\\ 54 \qquad 52|53\end{array}\right\rangle ::= \left\langle \begin{array}{l}\text{Arithme-}\\ \text{tischer}\\ \text{Ausdruck}\end{array}\right\rangle_{49} \left\langle \begin{array}{l}\text{Vergleichs-}\\ \text{operator}\end{array}\right\rangle_{13} \left\langle \begin{array}{l}\text{Arithme-}\\ \text{tischer}\\ \text{Ausdruck}\end{array}\right\rangle_{49} \Bigg|$$

$$\left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rischer}\\ \text{Ausdruck}\end{array}\right\rangle_{50} \left\langle \begin{array}{l}\text{Vergleichs-}\\ \text{operator}\end{array}\right\rangle_{13} \left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rischer}\\ \text{Ausdruck}\end{array}\right\rangle_{50} \Bigg|$$

$$\left\langle \begin{array}{l}\text{Arithme-}\\ \text{tischer}\\ \text{Ausdruck}\end{array}\right\rangle_{49}$$

zutrifft.

Hervorzuheben ist hierbei der Sonderfall, dass der komparative Ausdruck lediglich aus einem arithmetischen Ausdruck besteht. Diese Form wird implizit als eine Kurzform des komparativen Ausdrucks interpretiert, indem darunter der Vergleich

$$\langle \text{Arithmetischer Ausdruck} \rangle = 1$$

verstanden wird. Hat der arithmetische Ausdruck den Wert 1, so wird der komparative Ausdruck als wahr (true), andernfalls als falsch (false) angesehen. Nach ANSI BASIC ist diese implizite Kurzform des komparativen Ausdrucks nicht vorgesehen.[51a]
Denkbar wäre auch der Fall, dass der komparative Ausdruck lediglich aus einem alphanumerischen Ausdruck besteht; damit würde die Gleichung 54 zu folgender Beziehung erweitert:

$$\underset{54a}{\left\langle \begin{matrix}\text{Komparativer}\\ \text{Ausdruck}\end{matrix} \right\rangle} \underset{52|53}{:=} \underset{54}{\left\langle \begin{matrix}\text{Komparativer}\\ \text{Ausdruck}\end{matrix} \right\rangle} \,\Big|\, \underset{50}{\left\langle \begin{matrix}\text{Alphanumerischer}\\ \text{Ausdruck}\end{matrix} \right\rangle}$$

Der alleinstehende alphanumerische Ausdruck gilt implizit als alphanumerischer Vergleich der Art

$$\underset{50}{\left\langle \begin{matrix}\text{Alphanumerischer}\\ \text{Ausdruck}\end{matrix} \right\rangle} = \text{“TRUE”}$$

Enthält der alphanumerische Ausdruck den Stringinhalt mit der Zeichenfolge “TRUE”, so wird der Vergleich als korrekt (wahr) angesehen, andernfalls aber als unzutreffend interpretiert.
Generell gelangen komparative Ausdrücke nur in IF-Statements zur Anwendung, wie die folgenden Beispiele zeigen.

```
100  IF A$ = #DFUTSCH# THEN 200
110  IF B$ > C$ THEN 300
120  IF #JANUAR# <> D$ THEN 400
```

Besonders beachtenswert ist das zweite Statement, das einen Stringvergleich auf ASCII-Basis beinhaltet und aufgrund dessen etwa 'ABC' grösser als 'ABB', gleich 'ABC' und kleiner als 'ABD' oder 'ABCA' wäre.[52]

51a ANSI BASIC, p. 61–62.

52 Garland, Dartmouth BASIC, p. 27: “In the evaluation of string relations, the relation ‘less than’ is interpreted as meaning ‘earlier in alphabetical order than’, and the other relations are defined in similar ways. The alphabetical order of characters is defined by the ASCII code. ... If the two strings in a relational expression have different lengths, the shorter is extended by spaces on the right.” – Vgl. auch: Donald M. Monro. Interactive Computing with BASIC. A First Course. London 1974, p. 120.

Der FOSBIC-Compiler lässt die Verarbeitung logischer Ausdrücke nicht zu; er entspricht damit ANSI Minimal BASIC mit Standards für "numerical expressions" und "string expressions".[53]

325 Funktionen

Bei der Programmiersprache BASIC kann zwischen numerischen, alphanumerischen und hybriden Funktionen unterschieden werden.

⟨Funktion⟩ : = ⟨Numerische Funktion⟩ | ⟨Alphanumerische Funktion⟩ | ⟨Hybrid-funktion⟩
55 56 64 68a

Bei numerischen Funktionen sind Ergebnis und Argument rein numerisch, bei alphanumerischen Funktionen rein alphanumerisch. Die Hybridfunktionen unterscheiden sich von den rein numerischen resp. alphanumerischen Funktionen dadurch, dass sie einen Übergang zwischen numerischen und alphanumerischen Inhalten in dem Sinne herbeiführen können, dass bei numerischem Ergebnis das Argument alphanumerischen Inhalts und bei alphanumerischem Ergebnis numerischen Inhalts ist.
Bezüglich der Verwendung von Hybridfunktionen in arithmetischen (numerischen) und alphanumerischen Ausdrücken bestehen compilerspezifische Unterschiede. Üblicherweise sind Hybridfunktionen in arithmetischen Ausdrücken zugelassen, wenn sie zu einem numerischen Ergebnis führen. Entsprechendes gilt für Funktionen mit alphanumerischen Ergebnissen; sie können unabhängig davon, ob ihr Argument eventuell numerischen Inhalts ist, in alphanumerischen Ausdrücken benutzt werden. Hybridfunktionen sind nach ANSI Minimal BASIC nicht vorgesehen; bei ANSI BASIC werden sie unter den alphanumerischen Funktionen behandelt.[54]

325.1 Numerische Funktionen

Numerische Funktionen können in Form der eingebauten Funktionen (implementation-supplied numeric-functions) und als benutzerdefinierte Anweisungsfunktionen (user-defined numeric-functions) auftreten.

53 ANSI Minimal BASIC, p. 14–15.
54 ANSI BASIC, p. 39–41.

$$\langle \text{Numerische Funktion} \rangle_{56} ::= \langle \text{Eingebaute numerische Funktion} \rangle_{57} \mid \langle \text{Benutzerdefinierte numerische Funktion} \rangle_{62}$$

325.11 Eingebaute numerische Funktionen

Die eingebauten numerischen Funktionen stehen dem Benutzer über den BASIC-Prozessor jederzeit zur Verfügung, während die benutzerdefinierten Funktionen in den jeweiligen Programmen anwenderspezifisch vereinbart werden.
Nach ANSI Minimal BASIC sind nur elf Elementarfunktionen vorgesehen[55]; ANSI BASIC beinhaltet zusätzlich eine Reihe weiterer Funktionen und folgt damit der bisherigen praktischen Entwicklung von BASIC. Demnach sind eingebaute numerische Funktionen in Elementar- und Zusatzfunktionen zu unterteilen.

$$\langle \text{Eingebaute numerische Funktion} \rangle_{57} ::= \langle \text{Numerische Elementarfunktion} \rangle_{58} \mid \langle \text{Numerische Zusatzfunktion} \rangle_{60}$$

325.111 Elementarfunktionen

Numerische Elementarfunktionen (elementary numeric functions)[56] sind generell wie folgt strukturiert:

$$\langle \text{Numerische Elementarfunktion} \rangle_{58} ::= \langle \text{Elementarfunktionsname} \rangle_{59} \mid \langle \text{Elementarfunktionsname} \rangle_{59} \; (\langle \text{Arithmetischer Ausdruck} \rangle_{49}) \mid \langle \text{RANDOMIZE-Statement} \rangle_{59a}$$

$$\langle \text{Elementarfunktionsname} \rangle_{59} ::= \text{ABS} \mid \text{ATN} \mid \text{COS} \mid \text{EXP} \mid \text{INT} \mid \text{LOG} \mid \text{RND} \mid \text{SGN} \mid \text{SIN} \mid \text{SQR} \mid \text{TAN}$$

55 ANSI Minimal BASIC, p. 15.
56 Waite and Mather, BASIC, p. 32–34.

Nur RND kann ohne die (in Klammern beizufügende) Argumentangabe benutzt werden.
Im Rahmen der numerischen Elementarfunktionen nimmt RANDOMIZE insofern eine Sonderstellung ein, als diese Hilfsfunktion lediglich als alleinstehendes Statement aufgerufen werden darf.
Da ein arithmetischer Ausdruck auch eine numerische Funktion beinhalten darf, können Elementarfunktionen nicht nur einfach, sondern auch verschachtelt verarbeitet werden. Beispielsweise wird mit dem Programm

```
10  READ T
20  DATA 0.3
30  PRINT SIN(ATN(T))
40  END
```

der Sinus eines Winkels mit Tangens T berechnet und ausgedruckt.
Die Bedeutung der einzelnen eingebauten Elementarfunktionen ist aus Tabelle 4a ersichtlich.

Funktion	Bedeutung
ABS(X)	Absolutwert von X.
ATN(X)	Arcustangens von X in den Grenzen von $-(\pi/2) \leq ATN(X) \leq (\pi/2)$, wobei X im Bogenmass anzugeben ist.
COS(X)	Cosinus von X mit X im Bogenmass.
EXP(X)	e^x; für den Fall, dass der Wert e^x kleiner als die maschinenbedingte kleinste Zahl wird, sollte der Wert Null angenommen werden.
INT(X)	Grösste Ganzzahl, welche nicht grösser als X ist.
LOG(X)	Natürlicher Logarithmus von X.
RND	Pseudozufallszahl in den Grenzen $0 \leq RND \leq 1$.
SGN(X)	Algebraisches Vorzeichen von X; SGN(X) = − 1 für X kleiner Null, SGN(X) = 0 für X gleich Null, SGN(X) = 1 für X grösser Null.
SIN(X)	Sinus von X mit X im Bogenmass.
SQR(X)	Wurzel von X.
TAN(X)	Tangens von X mit X in Bogenmass.

Tabelle 4a. Eingebaute numerische Elementarfunktionen

RND ist nach ANSI Minimal BASIC ohne Argument zu verwenden; die meisten BASIC-Compiler erfordern jedoch noch die Angabe eines Scheinarguments.
Das Programm

```
100 FOR I = 1 TO 100
110 PRINT RND(A),
120 NEXT I
130 END
```

```
 0.4643      0.815013    0.442163    0.369334    0.665358
0.810336     0.961573    0.323004    0.675427    0.584752
0.634338     0.888465     0.50676    0.740409    0.820925
0.504587      0.19199    0.739858    0.964209    0.376601
0.168561      0.33723    0.331567    0.400685    0.403879
0.8763164    0.314105    0.268739   0.0502499    0.113351
0.634572     0.323632   0.0436654    0.747318    0.440532
0.860262     0.685499    0.165582    0.884348    0.692724
0.871173    0.0376757    0.416553    0.932907    0.254863
```

bewirkt die Ausgabe von hundert Zufallszahlen zwischen 0 und 1. Zufallszahlen zwischen 0 und 10 – Grenzwerte inbegriffen – erbringt das Programm

```
100 FOR I = 1 TO 100
110 PRINT INT(10 * RND(A)+0.5),
120 NEXT I
130 END
```

```
1          4          2          7          4
4         10          6          3          7
9          9          8          1          9
7          2          1          1          3
9          8          3          3          6
9        0.0          7         10          1
6          3          4          9          2
5          8          4          4          7
```

Die FOSBIC-Version 2 verfügt über eine interessante Variante der RND-Funktion, indem hier die Angabe eines Arguments folgende Ergebnisse erzeugt:

- Ist der Argumentwert gleich Null, so wird der maschineninterne Pseudozufallszahlengenerator auf seinen Ursprung zurückgesetzt, so dass eine mehrfache Wiederholung derselben Zufallszahlenreihe möglich wird.
- Ist der Argumentwert ungleich Null, so wird eine Pseudozufallszahlenreihe in den Grenzen

$$0 \leq \mathrm{RND}(X) \leq X, \text{ falls } X > 0 \text{ resp.}$$
$$X \leq \mathrm{RND}(X) \leq 0, \text{ falls } X < 0$$

erzeugt.

Mit dieser Variante der RND-Funktion kann demnach das obige Beispiel für Zufallszahlen zwischen 0 und 10 dahingehend vereinfacht werden, indem das Statement 110 durch die Anweisung

```
110 PRINT RND(10)
```

ersetzt wird.

Das RANDOMIZE-Statement ist auch in ANSI Minimal BASIC vorgesehen[56a] und weist die Form

$$\langle \text{RANDOMIZE-Statement} \rangle \; := \; \text{RANDOMIZE}$$
59a 58

56a ANSI Minimal BASIC, p. 25, 39.

auf. Durch seine Angabe wird der Pseudozufallszahlengenerator der RND-Funktion auf einen zufälligen Anfang gesetzt, so dass aus den Pseudozufallszahlen reale Zufallszahlenreihen entstehen, die nicht wiederholbar sind. Das RANDOMIZE-Statement ist nach der FOSBIC-Version 2 zulässig.

325.112 Zusatzfunktionen

Für die numerischen Zusatzfunktionen gilt generell

⟨Numerische Zusatzfunktion⟩ 60 57 : = ⟨Zusatzfunktionsname⟩ 61 | ⟨Zusatzfunktionsname⟩ 61 (⟨Numerische Aktualparameterliste⟩ 63)

⟨Zusatzfunktionsname⟩ 61 60 : =

ACOS | ASIN | ATN2 | CEIL | COSH | COT | CSC | DATE | DEG | EPS | FP | INF | IP | LOG10 | LOG2 | MAX | MIN | MOD | REM | PI | RAD | SEC | SINH | TANH | TIME | §§

Dabei seien mit §§ die compilerspezifisch verfügbaren numerischen Zusatzfunktionen bezeichnet.
Nur wenige Zusatzfunktionen können ohne Argumentangabe benutzt werden.
Die Bedeutung der nach ANSI BASIC vorgesehenen eingebauten numerischen Zusatzfunktionen ist aus Tabelle 4b ersichtlich.[56b]

56b ANSI BASIC, p. 23–25.

Funktion	Bedeutung
ACOS(X)	Arcuscosinus von X mit X im Bogenmass.
ASIN(X)	Arcussinus von X mit X im Bogenmass.
ATN2(X,Y)	Winkel im Bogenmass zwischen der positiven x-Achse und dem Vektor vom Nullpunkt zu den Koordinaten (X,Y); es gelten die Grenzwerte $-\pi < ATN2(X,Y) \leqq \pi$.
CEIL(X)	Ganzzahl, die nicht kleiner als X ist.
COSH(X)	Hyperbolischer Cosinus von X mit X im Bogenmass.
COT(X)	Cotangens von X mit X im Bogenmass.
CSC(X)	Cosekans von X mit X im Bogenmass.
DATE	Datum im Format YYDDD, mit YY = letzte beide Ziffern der Jahreszahl und DDD = Anzahl verflossene Tage innerhalb des Jahres.
DEG(X)	Umrechnung von X im Bogenmass nach Gradmass.
EPS	Kleinste positive maschinendarstellbare Zahl.
FP(X)	X − IP(X).
INF	Grösste positive maschinendarstellbare Zahl.
IP(X)	SGN(X) * INT(ABS(X)).
LOG10(X)	Dakadischer Logarithmus von X.
LOG2(X)	Binärer Logarithmus von X.
MAX(X,Y)	Maximum von X und Y.
MIN(X,Y)	Minimum von X und Y.
MOD(X,Y)	X − Y*INT(X/Y), wenn Y ungleich Null ist; falls Y = 0, wird der Funktionswert auch Null.
REM(X,Y)	X − Y*IP (X/Y) für Y ≠ 0; falls Y = 0, wird der Funktionswert auch Null.
PI	Konstante $\pi = 3{,}14159\ldots$
RAD(X)	Umrechnung von X im Gradmass nach Bogenmass.
SEC(X)	Sekans von X mit X im Bogenmass.
SINH(X)	Hyperbolischer Sinus von X mit X im Bogenmass.
TANH(X)	Hyperbolischer Tangens von X mit X im Bogenmass.
TIME	Zeit, Sekunden nach Mitternacht.

Tabelle 4b. Eingebaute numerische Zusatzfunktionen nach ANSI BASIC

Auch die FOSBIC-Version 2 verfügt über Zusatzfunktionen, die allerdings nur einen arithmetischen Ausdruck im Argument enthalten können. Die Kennwortstruktur entspricht nahezu vollständig derjenigen von ANSI BASIC. Ausserdem existieren noch die in Tabelle 4c aufgeführten eingebauten numerischen Zusatzfunktionen.

Funktion	Bedeutung
ACOT(X)	Arcuscotangens von X mit X im Bogenmass.
COTH(X)	Hyperbolischer Cotangens von X mit X im Bogenmass.
FLMAX	Grösster maschinenbedingter Exponent zur Basis 10 für Gleitkommazahlen.
INMAX	Grösster maschinenbedingter Exponent zur Basis 10 für Ganzzahlen.
FRA(X)	Dezimalteil von X.
ROU(X)	Kaufmännische Rundung von X auf zwei Dezimalstellen.
ARSIN(X)	Areafunktion Sinus von X.
ARCOS(X)	Areafunktion Cosinus von X; $X \geqq 1$.
ARTAN(X)	Areafunktion Tangens von X; $\lvert X \rvert < 1$.
ARCOT(X)	Areafunktion Cotangens von X; $\lvert X \rvert > 1$.

Tabelle 4c. Eingebaute numerische Zusatzfunktionen der FOSBIC-Version 2

Die numerischen Zusatzfunktionen dürfen beim FOSBIC-Compiler grundsätzlich nur Buchstaben enthalten, so dass für den dekadischen Logarithmus anstatt LOG10 der Funktionsname LOGD und für den binären Logarithmus LOGB angegeben werden muss. Weiterhin ist zu beachten, dass alle Funktionen stets ein Argument benötigen.

325.12 Benutzerdefinierte numerische Funktionen

Benutzerdefinierte numerische Funktionen sind nach ANSI Minimal BASIC für alle Compiler vorgesehen.[57] Sie werden durch DEF-Statements[58] formiert und gelangen im Programmablauf durch LET- oder PRINT-Statements zum Einsatz, wobei der Funktionsaufruf im einfachsten Fall in der Form

$$\text{LET} \left\langle \begin{matrix}\text{Numerische}\\ \text{Variable}\end{matrix} \right\rangle = \left\langle \begin{matrix}\text{Benutzerdefinierte}\\ \text{numerische Funktion}\end{matrix} \right\rangle$$

oder

$$\text{PRINT} \left\langle \begin{matrix}\text{Benutzerdefinierte}\\ \text{numerische Funktion}\end{matrix} \right\rangle$$

erfolgen kann.

57 ANSI Minimal BASIC, p. 24–25: "user-defined functions".
58 Vgl. Abs. 331.22.

Die Aktualisierung der Anweisungsfunktionen erfolgt in der Form

$$\underset{62}{\left\langle \begin{array}{l}\text{Benutzerdefinierte}\\ \text{numerische Funktion}\end{array}\right\rangle_{56}} := \text{FN}\ \underset{62a}{\langle\text{Funktionsname}\rangle}\left(\left\langle \begin{array}{l}\text{Numerische}\\ \text{Aktual-}\\ \text{parameterliste}\end{array}\right\rangle_{63}\right) \Bigg|$$

$$\text{FN}\ \underset{62a}{\langle\text{Funktionsname}\rangle}$$

mit

$$\underset{62a}{\langle\text{Funktionsname}\rangle} := \underset{62}{\langle\text{Grossbuchstabe}\rangle} \mid \underset{3a}{\langle\text{Grossbuchstabe}\rangle}\ \underset{5}{\langle\text{Ziffer}\rangle}$$

Nach ANSI Minimal BASIC ist als Funktionsname nur ein Grossbuchstabe zulässig[58a], nach ANSI BASIC jedoch auch die Kombination Grossbuchstabe gefolgt von einer Ziffer.[58b] Damit wird die bei den meisten BASIC-Compilern inzwischen eingetretene Entwicklung zum Standard erhoben.
Für die numerische Aktualparameterliste gilt:

$$\underset{63}{\left\langle \begin{array}{l}\text{Numerische}\\ \text{Aktual-}\\ \text{parameter-}\\ \text{liste}\end{array}\right\rangle} := \underset{49}{\left\langle \begin{array}{l}\text{Arith-}\\ \text{metischer}\\ \text{Ausdruck}\end{array}\right\rangle} \Bigg| \underset{63}{\left\langle \begin{array}{l}\text{Numerische}\\ \text{Aktual-}\\ \text{parameter-}\\ \text{liste}\end{array}\right\rangle}, \underset{49}{\left\langle \begin{array}{l}\text{Arith-}\\ \text{metischer}\\ \text{Ausdruck}\end{array}\right\rangle}$$

Aufgrund der vorstehenden Definitionsgleichung können theoretisch beliebig viele arithmetische Ausdrücke, durch Kommata getrennt, in der numerischen Aktualparameterliste aufgeführt werden. Bei Zusatzfunktionen ist der Listenumfang praktisch allerdings auf zwei arithmetische Ausdrücke beschränkt, und bei den benutzerdefinierten numerischen Funktionen erlauben nur wenige BASIC-Prozessoren mehr als zwei Ausdrücke.
Nach ANSI Minimal BASIC kann die numerische Aktualparameterliste nur einen, nach ANSI BASIC höchstens zwei arithmetische Ausdrücke umfassen.[59]
Die aktuellen Parameterwerte werden beim Aufruf einer benutzerdefi-

58a ANSI Minimal BASIC, p. 24.
58b ANSI BASIC, p. 73.
59 ANSI Minimal BASIC, p. 24; ANSI BASIC, p. 72–73.

nierten Funktion an die Stelle der formalen Parameter in der Definition der Benutzerfunktion übergeben und entsprechend in die Berechnung des Funktionswerts miteinbezogen.

325.2 Alphanumerische Funktionen

Auch bezüglich der alphanumerischen Funktionen ist zwischen eingebauten (implementation supplied) und benutzerdefinierten (user defined) Funktionen zu unterscheiden.[60]

$$\left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rische}\\ \text{Funktion}\end{array} \right\rangle_{64} :_{55} = \left\langle \begin{array}{l}\text{Eingebaute}\\ \text{alphanume-}\\ \text{rische}\\ \text{Funktion}\end{array} \right\rangle_{65} \Bigg| \left\langle \begin{array}{l}\text{Benutzer-}\\ \text{definierte}\\ \text{alphanume-}\\ \text{rische}\\ \text{Funktion}\end{array} \right\rangle_{68}$$

325.21 Eingebaute alphanumerische Funktionen

Analog zur Struktur der alphanumerischen Variablen charakterisieren sich alphanumerische Funktionen durch ein $-Zeichen im Funktionsnamen und im – nicht bei allen Funktionen explizit erforderlichen – Argument. Ergebnis und Argument sind alphanumerischen Inhalts, so dass für eingebaute alphanumerische Funktionen die Definition

$$\left\langle \begin{array}{l}\text{Eingebaute}\\ \text{alphanume-}\\ \text{rische}\\ \text{Funktion}\end{array} \right\rangle_{65} :_{64} = \left\langle \begin{array}{l}\text{String-}\\ \text{funktions-}\\ \text{name}\end{array} \right\rangle_{67} \Bigg| \left\langle \begin{array}{l}\text{String-}\\ \text{funktions-}\\ \text{name}\end{array} \right\rangle_{67} \left(\left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rische}\\ \text{Aktualpara-}\\ \text{meterliste}\end{array} \right\rangle_{66} \right)$$

gilt.
Die Aktualparameterliste kann mehrere, nach ANSI BASIC bis zu zwei, alphanumerische Ausdrücke umfassen.

$$\left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rische}\\ \text{Aktualpara-}\\ \text{meterliste}\end{array} \right\rangle_{66} :_{65} = \left\langle \begin{array}{l}\text{Alpha-}\\ \text{numerischer}\\ \text{Ausdruck}\end{array} \right\rangle_{51} \Bigg| \left\langle \begin{array}{l}\text{Alphanume-}\\ \text{rische}\\ \text{Aktualpara-}\\ \text{meterliste}\end{array} \right\rangle_{66} , \left\langle \begin{array}{l}\text{Alpha-}\\ \text{numerischer}\\ \text{Ausdruck}\end{array} \right\rangle_{51}$$

60 ANSI BASIC, p. 39, 72–73.

Unter besonderer Berücksichtigung der nach ANSI BASIC vorgesehenen Stringfunktionsnamen gilt:

$$\left\langle \begin{matrix}\text{String-}\\ \text{funktions-}\\ \text{name}\end{matrix} \right\rangle_{67,\,65} := \text{DATE\$} \mid \text{TIME\$} \mid \text{§§§§}$$

Dabei werden mit §§§§ die Namen weiterer compilerspezifisch verfügbarer Funktionen bezeichnet. Vgl. Tabelle 4d.

Funktion	Bedeutung
CHA$(A$,B$,C$)	Ersatz von String B$ in A$ durch C$.
DATE$	Datumausgabe in DD.MM.YY, wobei DD = Tag, MM = Monat, YY = Jahr.
DEL$(A$,B$)	Elimination von String B$ aus A$.
LEFT$(A$)	Linksbündige Ausrichtung von A$.
RIGHT$(A$)	Rechtsbündige Ausrichtung von A$.
TIME$	Tageszeit in HH.MM.SS, wobei HH = Stunden, MM = Minuten, SS = Sekunden.

Tabelle 4d. Eingebaute alphanumerische Funktionen

325.22 Benutzerdefinierte alphanumerische Funktionen

Problemspezifisch unter Verwendung von DEF-Statements definierte alphanumerische Funktionen werden via LET- oder PRINT-Statements aktiviert, wobei

$$\left\langle \begin{matrix}\text{Benutzer-}\\ \text{definierte}\\ \text{alphanume-}\\ \text{rische}\\ \text{Funktion}\end{matrix} \right\rangle_{68,\,64} := \text{FN}\ \langle\text{Funktionsname}\rangle_{62a}\ \$\ \left(\left\langle \begin{matrix}\text{Alphanumerische}\\ \text{Aktualparameter-}\\ \text{liste}\end{matrix} \right\rangle_{66}\right) \mid$$

$$\text{FN}\ \langle\text{Funktionsname}\rangle_{62a}\ \$$$

gilt.

Teilweise wird compilerspezifisch zur Funktionsbezeichnung nur ein Buchstabe und/oder als Aktualparameter nur ein alphanumerischer Aus-

druck zugelassen. Nach ANSI BASIC sind zwei Ausdrücke erlaubt, was auch für die FOSBIC-Version 2 zutrifft.

325.3 Hybridfunktionen

Die Hybridfunktionen sind dadurch gekennzeichnet, dass sie einen Übergang zwischen numerischen und alphanumerischen Inhalten herbeiführen können. Dies in dem Sinne, dass Funktionen mit alphanumerischem Argument ein numerisches Ergebnis und solche mit numerischem Argument ein alphanumerisches Ergebnis erbringen. Hybridfunktionen des ersten Typs werden als numerisch, diejenigen des zweiten Typs als alphanumerisch bezeichnet.

$$\underset{68a\quad 55}{\langle \text{Hybrid-funktion} \rangle} := \underset{68b}{\langle \text{Numerische Hybridfunktion} \rangle} \mid \underset{68c}{\langle \text{Alphanumerische Hybridfunktion} \rangle}$$

Die numerischen Hybridfunktionen können generell in arithmetischen Ausdrücken, die alphanumerischen Hybridfunktionen in alphanumerischen Ausdrücken verwendet werden.[61]

325.31 Numerische Hybridfunktionen

Die numerischen Hybridfunktionen lassen sich analog zu den rein numerischen Funktionen in eingebaute und benutzerdefinierte trennen.

$$\underset{68b\qquad 68a}{\langle \text{Numerische Hybridfunktion} \rangle} := \underset{68d}{\langle \text{Eingebaute numerische Hybridfunktion} \rangle} \mid \underset{68e}{\langle \text{Benutzerdefinierte numerische Hybridfunktion} \rangle}$$

325.311 Eingebaute numerische Hybridfunktionen

Für eingebaute numerische Hybridfunktionen gilt generell

$$\underset{68d\qquad 68b}{\langle \text{Eingebaute numerische Hybridfunktion} \rangle} := \underset{68f}{\langle \text{Numerischer Hybrid-funktions-name} \rangle} \; (\underset{68g}{\langle \text{Hybride alpha-numerische Aktualparameter-liste} \rangle})$$

61 Vgl. Abs. 324.1, 324.2.

mit

$$\left\langle \begin{array}{l}\text{Numerischer}\\ \text{Hybrid-}\\ \text{funktions-}\\ \text{name}\end{array}\right\rangle_{68f\ \ 68d} := \text{LEN} \mid \text{ORD} \mid \text{POS} \mid \text{VAL} \mid \S\S\S$$

Dabei seien mit §§§ Namen weiterer, compilerspezifisch verfügbarer, numerischer Hybridfunktionen bezeichnet.
Normalerweise besteht eine hybride alphanumerische Aktualparameterliste lediglich aus einem oder zwei alphanumerischen Ausdrücken. Ausnahmen bestehen bei den, nach ANSI BASIC vorgesehenen, Funktionen POS (A$, B$, M) und ORD (A$, M), mit denen durch den arithmetischen Ausdruck M eine Bearbeitung der Zeichenkette A$ ab dem m-ten Zeichen bewirkt wird.
Generell trifft mithin für die hybride alphanumerische Aktualparameterliste

$$\left\langle \begin{array}{l}\text{Hybride alphanumerische}\\ \text{Aktualparameterliste}\end{array}\right\rangle_{68g\ \ 68d} := \left\langle \begin{array}{l}\text{Alphanumerischer}\\ \text{Ausdruck}\end{array}\right\rangle_{50|51} \mid$$

$$\left\langle \begin{array}{l}\text{Arithmetischer}\\ \text{Ausdruck}\end{array}\right\rangle_{49}, \left\langle \begin{array}{l}\text{Hybride alphanumerische}\\ \text{Aktualparameterliste}\end{array}\right\rangle_{68g} \mid$$

$$\left\langle \begin{array}{l}\text{Hybride alphanumerische}\\ \text{Aktualparameterliste}\end{array}\right\rangle_{68g}, \left\langle \begin{array}{l}\text{Hybride alphanumerische}\\ \text{Aktualparameterliste}\end{array}\right\rangle_{68g}$$

zu.
Tabelle 4e zeigt nach ANSI BASIC vorgesehene eingebaute numerische Hybridfunktionen[62], die auch in der FOSBIC-Version 2 verfügbar sind.

Funktion	Bedeutung
LEN(A$)	Zeichenzahl in A$.
ORD(A$)	Ordinale Position des ersten Zeichens von A$ im compilerspezifischen Zeichensatz.
POS(A$,B$)	Position von B$ in A$.
VAL(A$)	Umwandlung von A$ in eine numerische Konstante.

Tabelle 4e. Eingebaute numerische Hybridfunktionen

62 ANSI BASIC, p. 39–40.

Vermerkt sei, dass Hybridfunktionen mit komplexen Aktualparameterlisten im allgemeinen durch Kombinationen einfacherer Funktionen ersetzt werden können. So lässt sich die Anweisung

```
100 X=ORD(A$,M)
```

durch die Anweisungsfolge

```
100 B$=A$(M:M)
110 X=ORD(B$)
```

ersetzen, falls nur die einfache eingebaute ORD-Funktion verfügbar ist und die Möglichkeit zur Teilzeichenkettenbegrenzung besteht.[63]

325.312 Benutzerdefinierte numerische Hybridfunktionen

Die benutzerdefinierten numerischen Hybridfunktionen unterscheiden sich von den rein numerischen Funktionen nur durch die hybride alphanumerische Aktualparameterliste, die in der Regel aus nur einem alphanumerischen Ausdruck besteht.

⟨Benutzerdefinierte numerische Hybridfunktion⟩ (68e, 68b) : = FN ⟨Funktionsname⟩ (62a) (⟨Hybride alphanumerische Aktualparameterliste⟩ (68g))

Benutzerdefinierte numerische Hybridfunktionen sind nach ANSI BASIC nicht vorgesehen.

325.32 Alphanumerische Hybridfunktionen

Alphanumerische Hybridfunktionen können eingebaut oder benutzerdefiniert sein.

⟨Alphanumerische Hybridfunktion⟩ (68c, 68a) : = ⟨Eingebaute alphanumerische Hybridfunktion⟩ (68h) | ⟨Benutzerdefinierte alphanumerische Hybridfunktion⟩ (68i)

63 Vgl. Abs. 323.23.

325.321 Eingebaute alphanumerische Hybridfunktionen

Eingebaute alphanumerische Hybridfunktionen weisen allgemein die Form

⟨Eingebaute alphanumerische Hybridfunktion⟩ (68h, 68c) : = ⟨Alphanumerischer Hybridfunktionsname⟩ (68j) (⟨Hybride numerische Aktualparameterliste⟩ (68k))

mit

⟨Alphanumerischer Hybridfunktionsname⟩ (68j, 68h) : = CHR$ | STR$ | §§§§

auf. Durch §§§§ werden Namen weiterer, compilerspezifisch eingebauter, alphanumerischer Hybridfunktionen bezeichnet.
Bezüglich der hybriden numerischen Aktualparameterliste gilt

⟨Hybride numerische Aktualparameterliste⟩ (68k, 68h) : = ⟨Arithmetischer Ausdruck⟩ (49) |

⟨Alphanumerischer Ausdruck⟩ (50|51), ⟨Hybride numerische Aktualparameterliste⟩ (68k) |

⟨Hybride numerische Aktualparameterliste⟩ (68k), ⟨Hybride numerische Aktualparameterliste⟩ (68k)

Bei den nach ANSI BASIC vorgesehenen Funktionen CHR$ und STR$ besteht die Aktualparameterliste nur aus einem arithmetischen Ausdruck[64]; vgl. Tabelle 4f.

Funktion	Bedeutung
CHR$(M)	m-tes Zeichen aus Liste der compilerspezifisch verfügbaren Zeichen.
STR$(X)	Stringumwandlung von X.

Tabelle 4f. Eingebaute alphanumerische Hybridfunktionen

325.322 Benutzerdefinierte alphanumerische Hybridfunktionen

Für benutzerdefinierte alphanumerische Hybridfunktionen gilt generell

$$\underset{68i}{\left\langle \begin{array}{l}\text{Benutzerdefinierte}\\ \text{alphanumerische}\\ \text{Hybridfunktion}\end{array}\right\rangle}_{68c} := \text{FN} \underset{62a}{\left\langle \begin{array}{l}\text{Funktions-}\\ \text{name}\end{array}\right\rangle} \$ \left(\underset{68k}{\left\langle \begin{array}{l}\text{Hybride numerische}\\ \text{Aktualparameter-}\\ \text{liste}\end{array}\right\rangle} \right)$$

Die hybride numerische Aktualparameterliste besteht in ihrer einfachsten Form nur aus einem arithmetischen Ausdruck. Nach ANSI BASIC ist diese Funktionsform noch nicht vorgesehen.

64 ANSI BASIC, p. 39–40.

Tabelle 5. Ausdrücke und eingebaute Funktionen

Prozessor	Arithmetischer Ausdruck[1]	Alphanumerischer Ausdruck[1]	Logischer Ausdruck[1]	Numerische Elementarfunktionen	Numerische Zusatzfunktion[2]	Alphanumerische Funktionen
ANSI Minimal BASIC	49	50	unzulässig	alle	keine	keine
Dartmouth BASIC-6	49	50/51	unzulässig	alle	15	6
UOM BASIC/ UBASIC	49	50/51	53	alle	26	9
IBM CALL BASIC	49	50	unzulässig	alle	18	keine
UWBIC	49 ohne 62	unzulässig	unzulässig	alle ausser SGN	keine	keine
FOSBIC	49 ohne 62	50	unzulässig	alle ausser SGN	keine	keine

[1] Die Zahlenangaben geben die Nummer der zulässigen Definitionsgleichung an.
[2] Die Zahlen geben die Anzahl der Zusatzfunktionen an.

BASIC ist eine zeilenorientierte Programmiersprache. Jedes BASIC-Programm besteht aus einer Folge von Programmzeilen oder -Sätzen, deren Charakter durch ein Kennwort bestimmt wird. Diese Programmsätze werden auch als Anweisungen oder Statements bezeichnet.

⟨Anweisung⟩ : = ⟨Statement⟩
69 76|205 70

Ein sinnvolles BASIC-Programm besteht mindestens aus zwei Anweisungen, von denen die letzte immer ein END-Statement ist[65]. Beispiel:

```
1  PRINT *EINFACHSTES PROGRAMM*
2  END
```

Die zu einem Programm gehörenden Anweisungen werden in der Regel sequentiell verarbeitet. Abweichungen von dieser Bearbeitungsregel werden durch Programmsteuerungsanweisungen[66] bewirkt. Insbesondere kann durch IF- und GOTO-Statements auf vorausgehende oder nicht unmittelbar folgende – durch Angabe der Statementnummer genau spezifizierte – Programmsätze übergegangen werden, wie das folgende Beispiel zeigt.

```
1      READ A, B
2      DATA 1, 1, 1, 2, 1, 3, 9999, 9999
3      IF A GE 9999 GOTO 9999
4      LET C = A + B
5      PRINT C
6      GOTO 1
9999   END
```

Zur Verwaltung von BASIC-Programmen auf den jeweiligen Maschinensystemen wurden vor allem für den Dialogbetrieb spezielle BASIC-Systemanweisungen entwickelt. So kann beispielsweise ein sich in Bearbeitung befindliches BASIC-Programm mittels einer entsprechenden BASIC-Systemanweisung auf einen externen Datenträger (Magnetband oder Magnetplatte) gerettet und später vom Zugriffsberechtigten mittels eines weiteren BASIC-Systembefehls von dort aufgerufen und in den Zentralspeicher der Maschine geladen werden.
BASIC-Systembefehle – sie werden ausführlich in Band 2 behandelt – dürfen nicht innerhalb von BASIC-Programmen erscheinen und haben deshalb auch keinerlei Auswirkungen auf die Logik des BASIC-Programmablaufs.

65 ANSI Minimal BASIC, p. 11; vgl. auch Abs. 14. – Garland, Dartmouth BASIC, p. 11; "BASIC is a line-oriented language. A BASIC program is a sequence of statements, each of which is identified by a key word. ...The end statement is required in BASIC since its absence may suggest a programming error."

66 Vgl. Abs. 332.4.

326.1 Statementstruktur

Die BASIC-Statements sind bei einer Programmverarbeitung im Dialogbetrieb[67] alle durchgehend zu numerieren; bei Monologbetrieb müssen nur die über Programmsteuerungsanweisungen zu erreichenden Bezugsstatements eine Nummer aufweisen. Statementnummern sind vorzeichenlose Ganzzahlen, grösser als Null.[68]

$$\langle \text{Statement} \rangle_{70} := \left\langle \begin{matrix}\text{Nicht-}\\ \text{numeriertes}\\ \text{Statement}\end{matrix} \right\rangle_{72} \Bigg| \left\langle \begin{matrix}\text{Statement-}\\ \text{nummer}\end{matrix} \right\rangle_{71} \left\langle \begin{matrix}\text{Nicht-}\\ \text{numeriertes}\\ \text{Statement}\end{matrix} \right\rangle_{72}$$

$$\left\langle \begin{matrix}\text{Statement-}\\ \text{nummer}\end{matrix} \right\rangle_{71} := \left\langle \begin{matrix}\text{Vorzeichenlose}\\ \text{Ganzzahl}>0\end{matrix} \right\rangle_{23}$$

Jedes nichtnumerierte BASIC-Statement besteht aus einem oder mehreren Kennworten (keywords) und seinem statementtypischen Inhalt.

$$\left\langle \begin{matrix}\text{Nicht-}\\ \text{numeriertes}\\ \text{Statement}\end{matrix} \right\rangle_{72} := \langle \text{Kennwort} \rangle_{73} \langle \text{Statementinhalt} \rangle_{75} \mid$$

$$\left\langle \begin{matrix}\text{Nicht-}\\ \text{numeriertes}\\ \text{Statement}\end{matrix} \right\rangle_{72} \langle \text{Kennwort} \rangle_{73} \left\langle \begin{matrix}\text{Statement-}\\ \text{inhalt}\end{matrix} \right\rangle_{75}$$

Aufgrund des Kenn- oder Befehlswortes wird vom BASIC-Compiler der entsprechende Operationscode zur Ausführung des Statements erzeugt und die dazugehörige Prüfliste zur Feststellung eventueller syntaktischer Statementfehler durchgangen. Jedes BASIC-Kennwort besteht aus einer vorgeschriebenen Folge von BASIC-Grundsymbolen. Im Hinblick auf die vorliegende Arbeit erweist es sich als zweckmässig, zwischen Elementar-, Sonder- und Systemkennworten zu unterscheiden.

67 Vgl. Abs. 11.

68 Einschränkend ANSI Minimal BASIC, p. 11: "Each line begins with a line-number, which does not contain and is not preceded by spaces. The value of the integer represented by a line-number must be nonzero; leading zeroes have no effect. Statements must occur in ascending line-number order."

⟨Kennwort⟩ : = ⟨Elementarkennwort⟩ | ⟨Sonderkennwort⟩ |
73 72 74 204

⟨Systemkennwort⟩
404

BASIC-Elementarkennworte dienen primär zur Bildung elementarer BASIC-Anweisungen (elementary statements), die bei allen BASIC-Prozessoren im wesentlichen die gleiche Struktur aufweisen.
Im Rahmen dieses BASIC-Bandes werden ausschliesslich Elementaranweisungen behandelt.
BASIC-Sonderkennworte werden zur Bildung von Spezial- oder Sonderanweisungen benötigt; diese dienen hauptsächlich zur (erleichterten) Behandlung von Spezialproblemen, wie etwa zur Matrizen- und Fileverarbeitung.
BASIC-Systemkennworte dienen zur Strukturierung von BASIC-Systemanweisungen, wie sie zur Verwaltung und Manipulation ganzer BASIC-Programme benötigt werden.
Für die nach ANSI Minimal BASIC explizit vorgesehenen Elementarkennworte (keywords) gilt[69]:

/Elementar-\
\kennwort/ : = BASE | DATA | DEF | DIM | END | FOR | GO |
74 73 GOSUB | GOTO | IF | INPUT | LET | NEXT |
ON | OPTION | PRINT | RANDOMIZE | READ |
REM | RESTORE | RETURN | STEP | STOP |
SUB | THEN | TO | §§§§

Die Zeichenfolge §§§§ bezeichnet Kennworte, die durch Sonderzeichen dargestellt (null-Statement, Image-Statement) oder im Zuge fortschreitender Standardisierung von BASIC zu Elementarkennworten erklärt werden. Mitenthalten ist auch das implizite data- und let-Statement.[70] Im Rahmen dieser Arbeit werden zusätzlich auch COMMENT, OF, PAGE, TAB und USING als Elementarkennworte betrachtet; die beiden letzterwähnten werden in Spezialformen des PRINT-Statements, die auch nach ANSI Minimal BASIC zugelassen sind, verwendet. SUB wird generell nicht als selbständiges Kennwort benutzt, sondern erscheint im Rahmen von ANSI Minimal BASIC nur in der Kennwortkombination GOSUB. Im allgemeinen werden die BASIC-Statements nach ihren Kennworten benannt.[71]

69 ANSI Minimal BASIC, p. 9.
70 Vgl. Abs. 332.12 und 332.32.
71 Vgl. Bd. 2, Abs. 32.

Kennwortfehler werden vom FOSBIC-Compiler generell mit folgendem Kommentar angezeigt:

```
YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

Der Statementinhalt besteht normaliter aus alleinstehenden oder sich folgenden, durch Trennzeichen voneinander getrennten BASIC-Ausdrükken. Ausnahmsweise kann der einem Kennwort folgende Statementinhalt auch aus Leerzeichen bestehen (PRINT-, PAGE-, STOP- und END-Statement).

$$\underset{75}{\langle\text{Statementinhalt}\rangle} \underset{72}{:=} \underset{7}{\langle\text{Leerzeichen}\rangle} \mid \underset{48}{\langle\text{Ausdruck}\rangle} \mid$$

$$\underset{48}{\langle\text{Ausdruck}\rangle} \underset{7}{\langle\text{Trennzeichen}\rangle} \mid$$

$$\underset{75}{\left\langle\begin{matrix}\text{Statement-}\\\text{inhalt}\end{matrix}\right\rangle} \underset{72}{\langle\text{Trennzeichen}\rangle} \underset{7}{\left\langle\begin{matrix}\text{Statement-}\\\text{inhalt}\end{matrix}\right\rangle} \underset{75}{}$$

Nach ANSI Minimal BASIC kann ein Statement bis zu 72 Zeichen umfassen.[72] Manche Compiler lassen beträchtlich längere BASIC-Statements zu; so beträgt die Längenbegrenzung bei Dartmouth BASIC-6 beispielsweise 132 Zeichen.[73] Vgl. Tabelle 6.
Beim FOSBIC-Compiler ist die Statementlänge auf 80 Zeichen (entsprechend der Lochkartenkapazität bzw. Bildschirmkapazität) begrenzt.

326.2 Statementnumerierung

Zur Statementnumerierung dürfen nur positive Ganzzahlen (>0) verwendet werden, deren Grösse bei den meisten BASIC-Compilern auf maximal vier oder fünf Ziffern begrenzt ist. Vgl. Tabelle 6.
Beim FOSBIC-Compiler wird die maximale Statementnummer bei der Compilerimplementierung eingestellt. Bei Überschreitung dieses Limits erfolgt die Fehlermeldung

```
SORRY - I ACCEPT ONLY LINE NUMBERS BETWEEN 1 AND  9999
SEE BASIC TEXTBOOK  A  326.2
```

72 ANSI Minimal BASIC, p. 11.
73 Garland, Dartmouth BASIC, p. 12.

Die Statements werden zweckmässigerweise fortlaufend mit einer Schrittweite von 10 numeriert, so dass nachträglich noch weitere Programmsätze sequentiell richtig in das Gesamtprogramm eingefügt werden können.
Nach ANSI Minimal BASIC müssen die einzelnen BASIC-Statements der Programmlogik entsprechend eingereiht und in aufsteigender Folge numeriert sein.[74]
Beim FOSBIC-Compiler ist sequentielle Statementnumerierung im Monologbetrieb nicht erforderlich; vorzugsweise werden – von Sonderfällen (Verwendung des DUMP-Statements[75], Austauschprogramme) abgesehen – nur die Bezugsstatements numeriert.
Im Dialogbetrieb kommt den Statementnummern insofern eine besondere Bedeutung zu, als die zu einem BASIC-Programm gehörenden Statements – unabhängig von der Eingabefolge – vor der Programmausführung in aufsteigender Reihenfolge sortiert werden.
Jede Statementnummer kann innerhalb eines Programms nur einmal zur Anweisungsnumerierung verwendet werden. Bei Mehrfachverwendung kommt es im Dialogbetrieb zur Elimination des früheren Statements gleicher Nummer.[76]
Beim FOSBIC-Compiler wird Mehrfachverwendung der gleichen Statenentnummer bei Monologbetrieb durch die Fehlermeldung

```
STATEMENT NUMBER    0 APPEARS MORE THAN ONCE
SEE BASIC TEXTBOOK  A  326.2
```

angezeigt.
Die Gesamtzahl der pro Programm zulässigen (numerierten) BASIC-Statements ist bei allen Compilern begrenzt. Diese Grenze wird bei Installation des FOSBIC-Compilers anlagenbezogen festgelegt; sie dürfte im allgemeinen über 2000 Statements pro Programm liegen.[77] Überschreitung der maximal zulässigen Statementzahl wird durch

```
COMPILATION STOPPED AT THIS POINT -- I CAN ONLY HANDLE
340 NUMBERED STATEMENTS
SEE BASIC TEXTBOOK  A  326.2
```

moniert.

74 ANSI Minimal BASIC, p. 11.
75 Vgl. Bd. 2, Abs. 333.21.
76 ANSI Minimal BASIC, p. 11 und Garland, Dartmouth BASIC, p. 12: "Local editing facilities may allow for the entry of statement lines in any order and also allow for duplicate line-numbers and lines containing only a line-number. Such editing facilities usually sort the program into the proper order. In the case of duplicate line-numbers, usually the last line entered with that line-number is retained. A line containing only a line-number (with no trailing spaces) is usually deleted from the program."

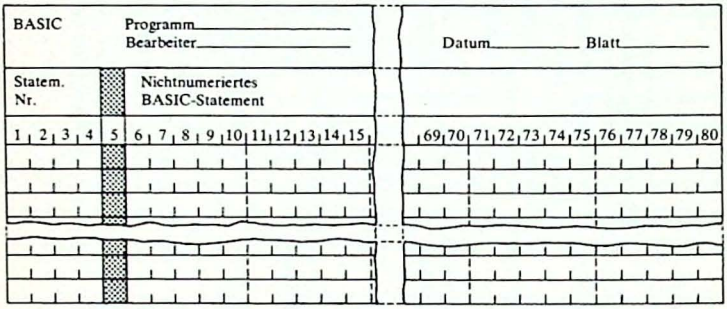
Tabelle 6. Länge und Numerierungsvorschriften für BASIC-Statements

Prozessor	Längen-begren-zung	maximale Statement länge	Numerierungsvorschrift				
			alle State-ments	sequen-tiell[1]	nur Bezugs-state-ments	höchste Statement-nummer	Anzahl nume-rierbarer Statements
ANSI Minimal BASIC	ja	72	ja	ja		9999	
Dartmouth BASIC-6	ja	132	ja	Sortierung		99999	
UOM BASIC/ UBASIC	ja	80	ja	ja		99999	500
IBM CALL BASIC	ja	255	ja	Sortierung		99999	1000
UWBIC	ja	80	nein	nein	ja	9999	500
FOSBIC	ja	80	nein	nein	ja	einstellbar	einstellbar

[1] ja = in steigend sequentieller Reihenfolge einzugeben. Sortierung = vor Compilierung in steigende Reihenfolge gebracht; Eingabe unsortiert möglich.

326.3 Codierblätter

Die Verwendung von Codierblättern drängt sich bei BASIC in geringerem Masse auf als bei anderen Programmiersprachen, ist aber aufgrund praktischer Erfahrungen zu empfehlen. Abbildung 18 zeigt einen entsprechenden Formularentwurf. Hierbei sind die ersten vier Spalten zur Aufnahme der Statementnummer und die einer Leerspalte folgenden Spalten für das nichtnumerierte BASIC-Statement vorgesehen.



bbildung 18. BASIC-Codierblatt

77 Vgl. Weber und Türschmann, FOSBIC-Compiler, S. 80.

33 Elementaranweisungen

Sämtliche BASIC-Anweisungen lassen sich – wie auch Abbildung 19 zeigt – in nichtausführbare und ausführbare Anweisungen einteilen, wobei bezüglich der ersteren weiterhin zwischen Kommentaren und Vereinbarungen zu differenzieren ist.

$$\underset{76}{\langle \text{Anweisung} \rangle} \underset{69}{:=} \left\langle \begin{matrix}\text{Nichtausführbare}\\ \text{Anweisung}\end{matrix} \right\rangle_{77} \mid \left\langle \begin{matrix}\text{Ausführbare}\\ \text{Anweisung}\end{matrix} \right\rangle_{78}$$

$$\left\langle \begin{matrix}\text{Nichtausführbare}\\ \text{Anweisung}\end{matrix} \right\rangle_{77} \underset{76}{:=} \underset{79}{\langle \text{Kommentar} \rangle} \mid \underset{85}{\langle \text{Vereinbarung} \rangle}$$

Die ausführbaren Anweisungen regeln die Dateneingabe und -Ausgabe, bewirken die Wertzuweisung an die Programmvariablen und steuern den Programmablauf. Sie lassen sich auch dementsprechend klassifizieren.

$$\left\langle \begin{matrix}\text{Ausführ-}\\ \text{bare An-}\\ \text{weisung}\end{matrix} \right\rangle_{78,\,76} := \left\langle \begin{matrix}\text{Eingabe-}\\ \text{anweisung}\end{matrix} \right\rangle_{109} \mid \left\langle \begin{matrix}\text{Wertzu-}\\ \text{ordnungs-}\\ \text{anweisung}\end{matrix} \right\rangle_{153} \mid \left\langle \begin{matrix}\text{Programm-}\\ \text{steuerungs-}\\ \text{anweisung}\end{matrix} \right\rangle_{164} \mid \left\langle \begin{matrix}\text{Ausgabe-}\\ \text{anwei-}\\ \text{sung}\end{matrix} \right\rangle_{122}$$

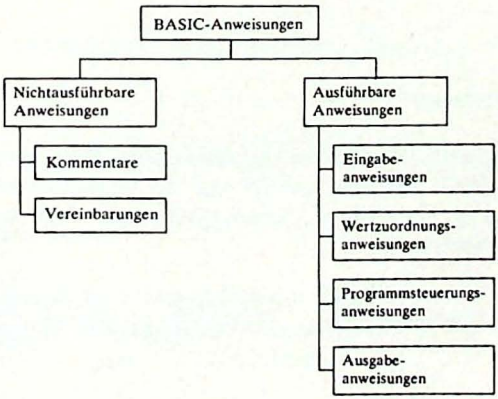


Abbildung 19.

Alle Anweisungen werden im folgenden als nichtnumerierte Statements dargestellt. Zudem werden in Band 1 nur sogenannte Elementaranweisungen behandelt; der entsprechende BASIC-Sprachumfang wird als Elementar-BASIC bezeichnet.[78]

331 Nichtausführbare Anweisungen

Wie bereits festgestellt wurde, ist bezüglich der nichtausführbaren Anweisungen zwischen Kommentaren und Vereinbarungen zu unterscheiden. Vgl. Abbildung 20. Während Kommentare den Programmablauf erläutern und zur Unterstützung des Programmverständnisses eingebaut werden, legen Vereinbarungen bestimmte Programmvoraussetzungen, wie maximale Feldgrössen von Matrizen oder benutzerdefinierte Funktionen, fest.

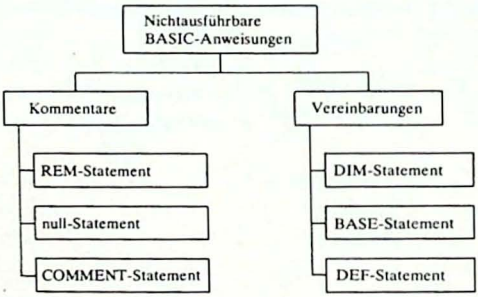


Abbildung 20.

331.1 Kommentare

Alle BASIC-Compiler bieten die Möglichkeit zur Programmkommentierung. Nach ANSI Minimal BASIC ist als Standardkommentar das REM-Statement vorgesehen. Alternativformen sind das null- und COMMENT-Statement.

$$\underset{79}{\langle \text{Kommentar} \rangle} \underset{77}{:=} \underset{80|81}{\langle \text{REM-Statement} \rangle} \mid \underset{82}{\langle \text{null-Statement} \rangle} \mid \underset{84}{\langle \text{COMMENT-Statement} \rangle}$$

78 Vgl. Nederlands Normalisatie-instituut. Proposal for ISO-Standard on programming Language Elementary BASIC. Rijswijk 1976.

331.11 REM-Statement

Das REM-Statement dient zur Vermittlung allgemeiner Informationen über ein Programm (Programmtitel, Autor, Programmierer, Programmversion), zur Kennzeichnung von Programmabschnitten, insbesondere auch von Unterprogrammen, zur Umschreibung der im Programm erscheinenden Eingabe- und Ausgabedaten und als Sprungziel, falls es mit einer Statementnummer versehen ist.
Das REM-Statement hat allgemein folgende Struktur:[79]

⟨REM-Statement⟩ : =
80 79

REM ⟨Alphanumerische Konstante⟩
32

Das REM-Statement kann an beliebiger Stelle in ein Programm eingefügt werden. Der Programmablauf selbst wird durch das Statement nicht beeinflusst. Es wird deshalb empfohlen, vom REM-Statement freizügig Gebrauch zu machen; dies insbesondere dann, wenn das Programm zu einem späteren Zeitpunkt erneut oder von Dritten benützt werden soll.

```
10 REM DIES IST EIN KOMMENTAR
```

Das REM-Statement lässt sich strukturkonform zum REMARK-Statement ausweiten.

⟨REM-Statement⟩ : =
81 79

REMARK ⟨Alphanumerische Konstante⟩
32

```
10 REMARK AUCH DIESE ZEICHENFOLGE WIRD ALS
20 REMARK/REM-STATEMENT INTERPRETIERT
```

331.12 null-Statement

Das null-Statement kann als verkürztes REM-Statement interpretiert werden; bei ihm tritt ein Sonderzeichen – im allgemeinen ' oder ! – an die Stelle von REM.

79 An Stelle des REM-Statements tritt bei Philips P350-S das NOTE-Statement. Vgl. Philips. BASIC 350-K Compiler für P350-S. Siegen 1974, S. 7.2.1.

⟨null-Statement⟩ : =
82 79

⟨null⟩ ⟨Alphanumerische Konstante⟩
83 32

⟨null⟩ : = '|!
83 82

null-Statements mit dem Sonderzeichen ' sind bei Dartmouth BASIC-6 zulässig[80]; das Ausrufzeichen kann bei EXBASIC anstelle von REM treten.[81] In ANSI BASIC wird diese Form des null-Statements als Standard vorgeschlagen.[82]
Der FOSBIC-Compiler lässt das null-Statement als verkürztes COMMENT-Statement zu, wobei das Ausrufzeichen (!) zur Kennung verwendet wird. Die Verwendung des null-Statements als Sprungziel ist nicht möglich. Druckausgabe erfolgt bei der CD 3300 in Form von √, was im folgenden bei einzelnen Beispielprogrammen zu beachten ist.
Einzelne Compiler lassen eine Verkettung des null-Statements mit anderen BASIC-Statements zu, womit allerdings das Prinzip der Zeilenorientiertheit von BASIC durchbrochen wird.[83]

```
10   IF X <= 0 THEN 30  ↓LOESUNG FUER X≤ NULL
```

331.13 COMMENT-Statement

Das COMMENT-Statement kann beim FOSBIC-Compiler anstelle des REM-Statements verwendet werden.

⟨COMMENT-Statement⟩ : =
84 79

COMMENT ⟨Alphanumerische Konstante⟩
32

80 Waite and Mather, BASIC, p. 35; Garland, Dartmouth BASIC, p. 44.
81 Com-Share Incorporated. NEWBASIC Reference Manual. Pittsburgh 1971, p. 15.
82 Vgl. ANSI BASIC, p. 15.
83 ANSI Minimal BASIC, p. 39: "The remark-statement should be the only means in this standard for including comments in a program. Enhancements may allow for comments at the end of lines, or for the null statement." – In ANSI BASIC (p. 15) wird dies bereits als Standard angegeben: "BASIC programs may be annotated by comments at the end of program lines or by separate remark-statements."

Im Gegensatz zum REM-Statement dient das COMMENT-Statement beim FOSBIC-Compiler nur zur Einfügung von Erläuterungen. Es kann eine Statementnummer erhalten, ist aber auch dann nicht als Bezugsstatement verwendbar.
Es empfiehlt sich, das COMMENT-Statement bei umfangreichen Erläuterungen einzusetzen, da der Compiler bei Anwendung dieses Statements keinen Speicherplatz zur Speicherung der Statementnummern verwendet.
Unkorrekte Schreibweise von COMMENT führt – wie auch beim REM-Statement – immer zur Fehlermeldung

```
YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

Prozessor	REM/ REMARK-Statement	COM-MENT-Statement	null-Statement als selbständiges Statement	null-Statement als Kommentar zu einem Statement
ANSI Minimal BASIC	v	n	n	n
Dartmouth BASIC-6	v	n	v	v
UOM BASIC/ UBASIC	v	n	v	v
IBM CALL BASIC	v	n	n	v
UWBIC	v	n	n	n
FOSBIC	v	v	v	v

v = verfügbar
n = nicht verfügbar

Tabelle 7. Verfügbarkeit der Kommentare

331.2 Vereinbarungen

Vereinbarungen sind nichtoperative Statements (descriptive statements) zur Definition von Feldern und Anweisungsfunktionen. Sie treten – vgl. Abbildung 20 – als DIM-, BASE- und DEF-Statements auf.

$$\underset{85}{\langle \text{Vereinbarung} \rangle} \underset{77}{:=} \underset{86}{\langle \text{DIM-Statement} \rangle} \mid \underset{97|98}{\langle \text{BASE-Statement} \rangle} \mid \underset{100}{\langle \text{DEF-Statement} \rangle}$$

ANSI BASIC zählt ausserdem noch das DATA-Statement und das IMAGE-Statement zu den Vereinbarungen.[83a] Dieser Neuzuordnung – sie wurde gegenüber ANSI Minimal BASIC lediglich aus formalen sprachsyntaktischen Gründen vorgenommen – wird im Rahmen dieser Arbeit nicht gefolgt.

331.21 DIM-Statements

DIM-Statements dienen zur expliziten Speicherplatzreservation für Felder.[84] Grundsätzlich ist zwischen numerischen, alphanumerischen und allgemeinen DIM-Statements zu unterscheiden.

$$\underset{86}{\langle \text{DIM-Statement} \rangle} \underset{85}{:=} \underset{91}{\langle \text{Numerisches DIM-Statement} \rangle} \mid \underset{94}{\langle \text{Alphanumerisches DIM-Statement} \rangle} \mid \underset{95}{\langle \text{Allgemeines DIM-Statement} \rangle}$$

DIM-Statements sind generell dann zu verwenden, wenn von der bei impliziter Dimensionierung standardmässig vorgegebenen Feldgrösse abzuweichen ist.[85]
DIM-Statements beinhalten pro Feld eine Deklaration des Feldnamens und der Feldgrösse. Diese Deklarationen weisen für numerische und alphanumerische Felder eine unterschiedliche Struktur auf.
Numerische Felder werden als Matrizen bezeichnet, wobei Vektoren als spezielle Matrizen mit nur einer Zeile resp. einer Spalte aufgefasst werden.[86] Analoges gilt für Textmatrizen bzw. -Vektoren, wobei allerdings die Elemente selbst wiederum aus einer Menge von Zeichen (alphanumerischen Konstanten) bestehen können.[86a]

83a ANSI BASIC, p. 45–46.
84 ANSI Minimal BASIC, p. 23: "The dimension-statement is used to reserve space for arrays."
85 Garland, Dartmouth BASIC, p. 42: "The dimension statement is used to reserve space for arrays if other than the standard space allocation is required." – Zur impliziten Dimensionierung vgl. Abs. 323.12.
86 Vgl. Heinz Körth, Carl Otto, Walter Runge und Manfred Schoch. Lehrbuch der Mathematik für Wirtschaftswissenschaften. Opladen 1972, S. 143–150.
86a Vgl. Abs. 322.2 und 323.2.

331.211 Numerische DIM-Statements

Für numerische Matrizen bzw. Vektoren ist die Deklarationsform

$$\begin{matrix}\langle\text{Numerische}\\ \text{Deklaration}\rangle\\ 87 \quad 91\end{matrix} : = \langle\text{Feldname}\rangle\ (\langle\text{Dimensionierung}\rangle) \mid \begin{matrix}\langle\text{Numerische}\\ \text{Deklaration}\rangle\\ 87\end{matrix}, \begin{matrix}\langle\text{Numerische}\\ \text{Deklaration}\rangle\\ 87\end{matrix}$$

$$40|41 \qquad 88$$

zu verwenden. Für den Feldnamen kann dabei abhängig vom vorhandenen BASIC-Compiler ein Buchstabe oder ein Alphazeichen eingesetzt werden.

$$\langle\text{Feldname}\rangle : = \langle\text{Buchstabe}\rangle$$
$$40 \quad 39 \qquad 3$$

$$\langle\text{Feldname}\rangle : = \langle\text{Alphazeichen}\rangle$$
$$41 \quad 39 \qquad 2$$

Nach ANSI BASIC ist auch die erweiterte Form des Feldnamens

$$\langle\text{Feldname}\rangle : = \begin{matrix}\langle\text{Gross-}\\ \text{buchstabe}\rangle\end{matrix} \mid \begin{matrix}\langle\text{Gross-}\\ \text{buchstabe}\rangle\end{matrix}\ \langle\text{Ziffer}\rangle$$
$$41a \quad 39 \qquad 3a \qquad 3a \qquad 5$$

zugelassen.[87]

Mit der Dimensionierung werden die Grenzen und die Art des Feldes (Vektor oder Matrix) festgelegt.

$$\langle\text{Dimensionierung}\rangle : = \langle\text{Zeilenzahl}\rangle \mid \langle\text{Spaltenzahl}\rangle \mid \langle\text{Zeilenzahl}\rangle,\ \langle\text{Spaltenzahl}\rangle$$
$$88 \qquad 87 \qquad 89 \qquad 90 \qquad 89 \qquad 90$$

Die Zeilen- oder Spaltenzahl muss eine vorzeichenlose Ganzzahl sein.

$$\langle\text{Zeilenzahl}\rangle : = \langle\text{Vorzeichenlose Ganzzahl}\rangle$$
$$89 \quad 88 \qquad 24$$

$$\langle\text{Spaltenzahl}\rangle : = \langle\text{Vorzeichenlose Ganzzahl}\rangle$$
$$90 \quad 88 \qquad 24$$

87 ANSI BASIC, p. 45–46.

Nach ANSI BASIC sollte bei einer Dimensionierung mit nur einer vorzeichenlosen Ganzzahl stets ein Spaltenvektor festgelegt werden. Diese Vereinbarung erlangt ihre besondere Bedeutung, wenn ein so dimensionierter Vektor in einer Matrizenanweisung verwendet wird.[88]
Das numerische DIM-Statement weist damit folgende Struktur auf:

⟨Numerisches DIM-Statement⟩ : =
91 86

DIM⟨Numerische Deklaration⟩ 87

Durch die DIM-Anweisung

```
10 DIM A(4), B(7,8), C(1,11)
```

werden drei Felder festgelegt, nämlich ein Spaltenvektor A mit 4 Spaltenelementen, eine Matrix B mit 7 Zeilen und 8 Spalten und ein Zeilenvektor C mit einer Zeile und 11 Spalten.
Weitere Dimensionierungsbeispiele und deren Anordnung im Hauptpeicher finden sich in Abs. 331.22.
Bei Grosscomputern kann die Dimensionierung von numerischen Variablen teilweise nicht nur mit zwei, sondern auch mit drei oder mehr Indizes erfolgen. So ist zum Beispiel beim Computersystem CD Cyber 174 das DIM-Statement

```
10 DIM A(5,6,7)
```

zulässig.[89]
Bei Kleincomputersystemen sind häufig weitere Varianten des numerischen DIM-Statements verfügbar. So kann bei einzelnen Systemen durch eine der numerischen Deklaration vorausgestellte Ziffer mit nachfolgendem %-Zeichen festgelegt werden, wie viele Maschinenworte als Speicherplatz für die numerischen Variablen verwendet werden sollen.[90]
Durch die DIM-Anweisung

```
10 DIM 1%,A, 2%,B(3), 4%,V(5,6)
```

wird für die einfache Variable A ein Speicherplatz von einem Wort fest-

88 Vgl. Bd. 2, Abs. 331.2.
89 Control Data. BASIC-Version 3. Reference Manual. Sunnyvale 1979, p. 2.2–2.
90 Vgl. etwa: Nixdorf. System-Software Business-BASIC. Paderborn 1979, S. 4.2–4.5.

gelegt; für jedes Element des Vektors B werden 2 Worte und für jedes Element der Matrix V 4 Worte vorgesehen. Dadurch können in den einzelnen numerischen Variablen jeweils unterschiedliche numerische Genauigkeiten erreicht werden.

331.212 Alphanumerische DIM-Statements

Für alphanumerische DIM-Statements gilt die Regel

```
⟨Alphanumerische⟩ : = ⟨Feldname⟩ $ (⟨Dimensionierung⟩) |
 ⟨Deklaration⟩                40|41                88
 92          94

                ⟨Feldname⟩ $ ⟨Maximale⟩ |
                   40|41       ⟨Stringlänge⟩
                                    93

                ⟨Feldname⟩ $ (⟨Dimensionierung⟩)
                   40|41                88

                ⟨maximale Stringlänge⟩ |
                                93

                ⟨Feldname⟩ $ ⟨Maximale⟩ (⟨Dimensionierung⟩) |
                   40|41       ⟨Stringlänge⟩                88
                                    93

                ⟨Alphanumerische⟩, ⟨Alphanumerische⟩ |
                 ⟨Deklaration⟩      ⟨Deklaration⟩
                        92                 92
```

mit

```
⟨maximale Stringlänge⟩ : = ⟨Vorzeichenlose Ganzzahl⟩ |
 93                92                      24

                        * ⟨Vorzeichenlose Ganzzahl⟩
                                            24
```

Das alphanumerische DIM-Statement weist damit folgende Struktur auf:

```
⟨Alphanumerisches DIM-Statement⟩ : =
 94                            86

 ┌──────────────────────────────────┐
 │ DIM ⟨Alphanumerische Deklaration⟩ │
 │                            92    │
 └──────────────────────────────────┘
```

Die alphanumerischen Deklarationen unterscheiden sich von den numerischen zunächst wiederum nur durch das $-Zeichen. Zusätzlich kann jedoch im alphanumerischen DIM-Statement in jeder Deklaration die maximale Stringlänge festgelegt werden, die die alphanumerische Variable aufnehmen kann. Handelt es sich dabei um eine indizierte Variable, so gilt die Vorgabe der maximalen Stringlänge für jedes Element. Fehlt die Angabe der maximalen Stringlänge, so wird implizit eine compilerspezifisch vorgegebene maximale Stringlänge angenommen. Nach ANSI BASIC sollte diese nicht weniger als 72 Zeichen umfassen[91]; bei der FOSBIC-Version 2 sind dies 9 Zeichen.
Nach ANSI BASIC muss die Spezifikation der maximalen Stringlänge der Dimensionierung nachgestellt und stets mit einem *-Zeichen versehen werden.[92] Die FOSBIC-Version 2 verarbeitet alle in Gleichung 92 angegebenen Formen.
In alphanumerischen DIM-Statements können einfache alphanumerische Variable dann auftreten, wenn ihre maximale Stringlänge von der compilerspezifisch impliziten Festlegung (bei WANG 2200 beispielsweise 16 Zeichen) abweichen soll.

```
5  REM WANG-BEISPIEL
10 DIM A$, B$7, C$23
20 READ A$, B$, C$,
30 DATA ≠ABCDEFGHIJKLMNOPQRSTUVWXYZ≠
40 DATA ≠ABCDEFGHIJKLMNOPQRSTUVWXYZ≠
50 DATA ≠ABCDEFGHIJKLMNOPQRSTUVWXYZ≠
60 PRINT A$
70 PRINT B$
80 PRINT C$
90 END

   ABCDEFGHIJKLMNOP
   ABCDEFG
   ABCDEFGHIJKLMNOPQRSTUVW
```

Der FOSBIC-Compiler lässt die letzterwähnte Art der Felddeklaration noch nicht zu. Für das vorstehende Programm müsste das FOSBIC-konforme Programm wie folgt aufgebaut sein:

```
10  DIM A$(4),B$(2),C$(5)
20  READ A$(1),A$(2),A$(3),A$(4)
30  READ B$(1),B$(2)
40  READ C$(1),C$(2),C$(3),C$(4),C$(5)
50  DATA ≠ABCDEFGHIJKLMNOP≠
60  DATA ≠ABCDEFG≠
70  DATA ≠ABCDEFGHIJKLMNOPQRSTUVW≠
80  PRINT A$(1);A$(2);A$(3);A$(4)
90  PRINT B$(1);B$(2)
100 PRINT C$(1);C$(2);C$(3);C$(4);C$(5)
110 END


ABCDEFGHIJKLMNOP
ABCDEFG
ABCDEFGHIJKLMNOPQRSTUVW
```

91 ANSI BASIC, p. 43, 47. – Das alphanumerische DIM-Statement ist in ANSI Minimal BASIC noch nicht vorgesehen.
92 ANSI BASIC, p. 43.

Die Strings in den DATA-Statements müssen dazu entsprechend festgelegt werden.[93]

331.213 Allgemeine DIM-Statements

Numerische und alphanumerische Deklaration können im allgemeinen in beliebiger Reihenfolge in einem DIM-Statement angegeben werden. Für dieses allgemeine DIM-Statement gilt:

⟨Allgemeines DIM-Statement⟩ : =
95 86

DIM ⟨Deklaration⟩
96

⟨Deklaration⟩ : = ⟨Numerische Deklaration⟩ | ⟨Alphanumerische Deklaration⟩ |
96 95 87 92

⟨Deklaration⟩, ⟨Deklaration⟩
96 96

Demnach könnten die folgenden einzelnen numerischen und alphanumerischen DIM-Statements

```
10  DIM A(20)
20  DIM B(1,3)
30  DIM C$(8)
40  DIM S(4,4)
```

wie folgt zusammengefasst werden

```
10  DIM A(20), B(1,3), C$(8), S(4,4)
```

In einem mehrfachen DIM-Statement kann die Reihenfolge der Deklarationen frei gewählt werden, wie das folgende Beispiel (FOSBIC-Version 2) zeigt.

```
                  R$* 20(1),
55 DIM R$*5(8), A$(45), B$(6,7)*22, C(5,5)
```

Mit diesem DIM-Statement wird ein alphanumerischer Spaltenvektor R$ mit 8 Spaltenelementen und einer Stringlänge von 5 Zeichen je Element, eine alphanumerische Variable A$ mit einer maximalen Stringlänge von 45 Zeichen, eine alphanumerische Matrix B$ mit 6 Zeilen, 7 Spalten und einer Stringlänge von 22 Zeichen je Element und eine numerische Matrix C mit 5 Zeilen und 5 Spalten festgelegt.

93 Vgl. Abs. 332.11.

In Verbindung mit dem DIM-Statement auftretende Fehler werden vom FOSBIC-Compiler detailliert ausgewiesen.
Kennwortfehler bei DIM:

```
        YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

Unzulässiger Feldname, Wegfall von rechter oder/und linker Klammer bei der Dimensionierung:

```
        DIM STATEMENT INCORRECTLY WRITTEN
SEE BASIC TEXTBOOK  A 331.21
```

Unzulässige Zeilen- oder Spaltenzahl bzw. Überschreiten des verfügbaren Speicherplatzes:

```
**** YOU ARE TOO GREEDY -- YOUR PROGRAM PLUS DATA REQUIRES
     16982 MORE LOCATIONS THAN I CAN GIVE YOU
```

bzw.

```
        PROGRAM AND DATA EXCEED AVAILABLE STORAGE AT THIS POINT
SEE BASIC TEXTBOOK  A  331.2
```

Mehrfache Dimensionierung der gleichen Variablen führt zur Warnung

```
        VARIABLE 0 REDIMENSIONED -- BUT I TAKE IT AS A  JOKE
SEE BASIC TEXTBOOK  A 331.21
```

Tritt bei impliziter Dimensionierung die Meldung

```
PROGRAM STOPPED -- THERE IS NOT ENOUGH SPACE FOR ITEM NUMBER
                   12 IN LIST OR TABLE A
                   ERROR CODE= 17
              ***** SEE BASIC TEXTBOOK =  A 331.21  *****
              ***** I WAS AT LINE NUMBER  0
```

auf, so reichte die standardmässig vorgegebene Speicherplatzreservation zur Programmbearbeitung nicht aus; es erfolgt Programmabbruch zur Ermöglichung der notwendigen expliziten Dimensionierung.

331.22 BASE-Statement

Nach ANSI Minimal BASIC lautet die Definition des BASE-Statements[94]

94 ANSI Minimal BASIC, p. 23.

⟨BASE-Statement⟩ : =
97 85

OPTION BASE ⟨Basis⟩
99

Nach ANSI BASIC entfällt das Kennwort OPTION und es gilt [94a]

⟨BASE-Statement⟩ : =
98 85

BASE ⟨Basis⟩
99

mit

⟨Basis ⟩ : = 0 | 1
99 97|98

Durch das BASE-Statement wird die Untergrenze aller in Dimensionierungen auftretender indizierter Variabler festgelegt. Aus diesem Grunde muss das BASE-Statement vor dem ersten DIM-Statement bzw. vor der ersten impliziten Festlegung einer indizierten Variablen im Programm aufgerufen werden. Sinnvollerweise stellt man daher dieses Statement an den Programmanfang.
Normalerweise werden bei der Dimensionierung einer nm-Matrix die Zeilen- und Spaltenzahlen mit n und m angegeben.
Bei Felddimensionierungen auf der Basis von 0 oder 1 sind auch Statements der Art DIM A(0) oder DIM A(0,0) möglich; sie sind für den gleichen Feldnamen nur alternativ verwendbar und führen zur Reservation eines einzigen Speicherpatzes für die indizierte numerische Variable A(0) resp. A(0,0). Mit DIM A(1,1) werden dagegen – vgl. Abbildung 21 – für vier Elemente der Matrix A, nämlich für A(0,0), A(0,1), A(1,0) und A(1,1), Speicherplätze reserviert. Demgegenüber wird bei Felddimensionierung auf der Basis von 1 mit dem Statement DIM A(1,1) nur ein Speicherplatz für A(1,1) freigehalten.
Bei Dimensionierung mit der Basis 0 wird – wie aus Tabelle 8 ersichtlich ist[94b] – insbesondere bei kleinen Matrizen ceteris paribus wesentlich mehr Speicherplatz reserviert als bei Dimensionierung auf der Basis von 1.

94a ANSI BASIC, p. 46.
94b Vgl. Jerome C. Kraus. Array Origin Impass. X3J2 Document 75/04, p. 2.

Beispiel	Anzahl Speicherplätze Basis 0	Basis 1	Differenz zur Basis 0
DIM A (100, 100)	10201	10000	2,1 %
DIM B (10, 10)	121	100	21 %
DIM C (5, 5)	36	25	36 %
DIM D (1, 1)	4	1	300 %

Tabelle 8. Speicherplatzreservation auf der Basis von 0 und 1

Beispielsweise bewirken die DIM-Statements DIM A(200) und DIM B(20,20), dass für den Vektor A normaliter 201 Speicherplätze und für die Matrix B 441 Speicherplätze, zusammen mithin 642 Speicherplätze reserviert werden. Durch explizite Dimensionierung auf der Basis von 1 könnte die Speicherplatzreservation auf 600 reduziert werden.
Bezogen auf das letzterwähnte Beispiel kann nach ANSI Minimal BASIC die Dimensionierung mit der Basis 0 auch explizit in der Form DIM A(0:200) und DIM B(0:20,0:20) vorgenommen werden; zur speicherplatzsparenden Dimensionierung auf der Basis von 1 wäre DIM A(1:200) resp. DIM B(1:20,1:20) erforderlich.
Die meisten BASIC-Compiler lassen die beschriebene Optionsmöglichkeit nicht zu; so erfolgt auch beim FOSBIC-Compiler die Dimensionierung generell mit der Basis 0.
Bei kleinen Matrizen sind Speicherplatzeinsparungen generell dadurch erreichbar, dass die bei impliziter Dimensionierung standardmässig vorgegebenen Feldgrössen explizit – via DIM-Statement – auf die problemspezifisch erforderliche Minimalgrösse reduziert werden.
Vom FOSBIC-Compiler werden bei impliziter Dimensionierung für eine Matrix stets 121 Speicherplätze reserviert.[94c] Mithin werden beispielsweise mit dem Statement LET A(I,J) = B(I) *C(J) bei erstmaligem Erscheinen der Variablen B(I), C(J), A(I,J) im Quellenprogramm implizit 363 Speicherplätze belegt. Ceteris paribus könnten durch ein vorausgehendes DIM-Statement der Form DIM A(4,4), B(4), C(4) mehr als 300 Speicherplätze eingespart werden.
Die Zuweisung der reservierten Speicherplätze an bestimmte Matrixelemente ist compilerabhängig. Bei Dimensionierung auf der Basis von 0 und zeilenweiser Anordnung der Elemente wird dem Element a_{ij} der Matrix A_{mn} der Speicherplatz $SZ_{ij} = (n+1)\,i + (j+1)$ zugewiesen, bei spaltenweiser Anordnung der Speicherplatz $SS_{ij} = (i+1) + (m+1)j$; vgl. Abbildung 21.

94c ANSI Minimal BASIC, p. 23: "The default space allocation reserves space for 11 elements in one-dimensional arrays and 121 elements in two-dimensional arrays."

i \ j	0	1	...	n
0	$a_{o,o}$ SZ = 1 SS = 1	$a_{o,1}$ SZ = 2 SS = m + 2	...	$a_{o,n}$ SZ = n + 1 SS = (m + 1) n + 1
1	$a_{1,o}$ SZ = n + 2 SS = 2	.	...	$a_{1,n}$ SZ = 2 · (n + 1) SS = (m + 1) n + 2
⋮	⋮	⋮	...	⋮
m	$a_{m,o}$ SZ = (n + 1) m + 1 SS = m + 1	$a_{m,1}$ SZ = (n + 1) m + 2 SS = 2 · (m + 1)	...	$a_{m,n}$ SZ = (m + 1) (n + 1) SS = (m + 1) (n + 1)

Abbildung 21. Speicherplatzreservation für ein- oder zweidimensionale Felder A(I) resp. A(I,J)

Im allgemeinen erfolgt die Speicherplatzzuweisung zeilenweise, was bei der Matrizenverarbeitung zu beachten ist.[95] Für den Anfänger empfiehlt sich deshalb, die Matrizenverarbeitung generell zeilenweise zu programmieren.

Beispielsweise ist die zeilenweise Addition der folgenden Matrizen

$$C = A + B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} + \begin{bmatrix} 12 & 11 & 10 & 9 \\ 8 & 7 & 6 & 5 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 13 & 13 & 13 & 13 \\ 13 & 13 & 13 & 13 \\ 13 & 13 & 13 & 13 \end{bmatrix}$$

FOSBIC-konform in der Form

```
10  DIM A(3,4),B(3,4),C(3,4)
15  PRINT ≠          I≠,≠          J≠,≠     A(I,J)≠,≠     B(I,J)≠,≠     C(I,J)≠
20  FOR I=1 TO 3
30    FOR J=1 TO 4
40      READ A(I,J),B(I,J)
50      LET C(I,J)=A(I,J)+B(I,J)
60      PRINT I,J,A(I,J),B(I,J),C(I,J)
70    NEXT J
80  NEXT I
90  DATA 1,12,2,11,3,10,4,9,5,8,6,7
100 DATA 7,6,8,5,9,4,10,3,11,2,12,1
110 END
```

95 Vgl. Bd. 2, Abs. 311.111.

I	J	A(I,J)	B(I,J)	C(I,J)
1	1	1	12	13
1	2	2	11	13
1	3	3	10	13
1	4	4	9	13
2	1	5	8	13
2	2	6	7	13
2	3	7	6	13
2	4	8	5	13
3	1	9	4	13
3	2	10	3	13
3	3	11	2	13
3	4	12	1	13

zu programmieren. Das Beispiel zeigt auch, dass explizite Matrizendimensionierung die Programmkontrolle zu erleichtern vermag; so müssen A und B identische Feldgrössen aufweisen, während C mindestens gleich A und B dimensioniert sein muss.

Tabelle 9. Felddimensionierung

Prozessor	Numerische Felder[1]				Alphanumerische Felder[1]			
	explizit		implizit		explizit		implizit	
	Vektor max. Elem.	Matrix max. Elem.	Vektor max. Elem.	Matrix max. Elem.	Vektor max. Elem.	Matrix max. Elem.	Vektor max. Elem.	Matrix max. Elem.
ANSI Minimal BASIC			121	11 × 11		unzulässig		
Dartmouth BASIC-6	2^{18}	2^{18}	11	11 × 11	2^{18}	2^{18}	11	11 × 11
UOM BASIC/ UBASIC	4000	4000	11	11 × 11		unzulässig	11	unzulässig
IBM CALL BASIC	7679/ 15359	7679/ 15359	10	10 × 10	3413	unzulässig	10	unzulässig
UWBIC	einstellbar		121	11 × 11	unzulässig			
FOSBIC allgemein	einstellbar		121	11 × 11	einstellbar		121	11 × 11
CD 3300 bei 64 qp u. Doppelwortoption	3700	3700	121	11 × 11	3700	3700	121	11 × 11

[1] Die Zahlenangaben beziehen sich auf die maximal dimensionierbare Elementzahl.

Fortsetzung der Tabelle 9. Felddimensionierung

Prozessor	Index Null zulässig	Anzahl der maximalen Indizes numerisch	alphanum.	Zeichenkettenlänge je Element in alphanum. Feldern festlegbar
ANSI Minimal BASIC	ja	2	unzulässig	nein
Dartmouth BASIC-6	ja	2	2	nein
UOM BASIC/ UBASIC	ja	4	1	ja
IBM CALL BASIC	nein	2	1	nein
UWBIC	ja	2	unzulässig	unzulässig
FOSBIC allgemein	ja	2	2	nein
CD 3300 bei 64 qp und Doppelwort-option	ja	2	2	nein

331.23 DEF-Statements

DEF-Statements dienen zur Vereinbarung benutzerdefinierter numerischer und alphanumerischer Funktionen (user-defined functions).[95a]

⟨DEF-Statement⟩ : = ⟨Numerisches DEF-Statement⟩ | ⟨Alpha-numerisches DEF-Statement⟩ |
100 85 101|104 105|108

⟨Hybrides DEF-Statement⟩
108a

Die FOSBIC-Version 2 lässt DEF-Statements zu.

331.231 Numerische DEF-Statements

Numerische DEF-Statements können in einfacher und – bei manchen BASIC-Compilern – auch in ausgeweiteter Form gebildet werden[96]; dementsprechend wird in der angloamerikanischen Literatur zwischen "one line DEF statements" und "multiple line DEF statements" unterschieden.[97]
Nach ANSI Minimal BASIC sind nur einzeilige Anweisungsfunktionen vorgesehen, die in zwei Alternativformen realisierbar sind.

⟨Numerisches DEF-Statement⟩ : =
101 100

DEF FN	⟨Numerischer Funktions-identifikator⟩ 102	=	⟨Arithmetischer Ausdruck⟩ 49

95a Vgl. Abs. 325.12, 325.22, 325.12, 325.322.

96 "[User-defined functions] may be simple one-line definitions in terms of an expression, or they may be more complicated multiple-line definitions." Bull, Freeman and Garland, Specification for Standard BASIC, p. 70.

97 Waite and Mather, BASIC, p. 63–65. – Garland (Dartmouth BASIC, p. 39) verwendet die Bezeichnungen "simple-def-line" und "extended-def-block".

$$\langle \text{Numerischer Funktions-identifikator} \rangle_{102} ::= \langle \text{Buchstabe} \rangle_{3} \mid \langle \text{Buchstabe} \rangle_{3} \; (\langle \text{Numerische Formalparameterliste} \rangle_{103})$$

Nach ANSI BASIC können vom Benutzer mehr als 26 numerische Funktionen definiert werden, da die Beziehung

$$\langle \text{Numerischer Funktionsindikator} \rangle_{102a} ::= \langle \text{Funktionsname} \rangle_{62b} \mid \langle \text{Funktionsname} \rangle_{62b} \; (\langle \text{Numerische Formalparameterliste} \rangle_{103})$$

zulässig ist.[97a]
Die numerische Formalparameterliste entspricht in ihrer Struktur der numerischen Aktualparameterliste (vgl. Gleichung 63, Abs. 325.12), wobei anstelle der arithmetischen Ausdrücke jetzt allerdings nur einfache numerische Variable verwendet werden dürfen.

$$\langle \text{Numerische Formalparameterliste} \rangle_{103} ::= \langle \text{Einfache numerische Variable} \rangle_{37} \mid \langle \text{Numerische Formalparameterliste} \rangle_{103} \, , \, \langle \text{Einfache numerische Variable} \rangle_{37}$$

Einzeilige Anweisungsfunktionen können somit generell ohne oder mit Formalparameterliste gebildet werden, wie auch die Beispiele DEF FNA = A(I) und DEF FNB(X,Y) = SQR(X*X + Y*Y) zeigen. DEF-Statements ohne Formalparameter können insbesondere auch zur Definition häufig verwendeter Konstanter benutzt werden; so kann etwa π durch DEF FNP = 3.14159 in programmrelevanter Genauigkeit festgelegt werden.
In der Regel enthalten die Anweisungsfunktionen Formalparameter (dummy variables). Diese Pseudovariablen können – unter Einhaltung der Regeln zur Bildung einfacher numerischer Variabler – beliebig gewählt werden; sie können insbesondere auch einer an anderer Stelle im

97a ANSI BASIC, p. 73.

Programm verwendeten einfachen numerischen Variablen entsprechen, ohne dass dieser Doppelgebrauch vom Compiler moniert würde.[98]
Nach ANSI/ECMA Minimal BASIC darf die Zahl der Formalparameter höchstens 1 betragen.[99] Die meisten BASIC-Compiler lassen allerdings zwei oder mehr Formalparameter zu; vgl. Tabelle 10. Diese Erweiterung wird durch ANSI BASIC zum Standard erhoben.
Der zu verarbeitende arithmetische Ausdruck kann beliebige Formalparameter enthalten (die vor dem Funktionsaufruf zu aktualisieren sind); zulässig sind auch Standard- und Anweisungsfunktionen.[100]
Vermerkt sei auch, dass jedes DEF-Statement an beliebiger Stelle des Programms – vor oder nach Gebrauch der Anweisungsfunktion – erscheinen kann; es empfiehlt sich aber, die Anweisungsfunktionen an den Programmanfang zu stellen.
Jede Funktion darf im gleichen Programm nur einmal definiert werden.
Das folgende Beispiel zeigt die Einsatzmöglichkeit von Anweisungsfunktionen zur Lösung von Gleichungen zweiten Grades.[101]

```
 10  REM GLEICHUNG ZWEITEN GRADES ALS ANWEISUNGSFUNKTIONEN
 20  DEF FND(A,B,C) = B↑2-4*A*C
 30  DEF FNX(A,B,C) = (-B+SQR(FND(A,B,C)))/(2*A)
 40  DEF FNY(A,B,C) = (-B-SQR(FND(A,B,C)))/(2*A)
 50  DEF FNZ(B,C) = -C/B
 60  PRINT ≠       A≠,≠       B≠,≠       C≠,≠       X(1)≠,≠       X(2)≠,≠BEMERKUNG≠
 70  READ A,B,C
 80  IF A <> 0 THEN 120
 90  IF B <> 0 THEN 150
100  IF C <> 0 THEN 190
110  GOTO 900
120  IF FND(A,B,C)<0 THEN 170
130  PRINT A,B,C,FNX(A,B,C),FNY(A,B,C),≠ REELL≠
140  GOTO 70
150  PRINT A,B,C,FNZ(B,C),≠ ≠,≠ LINEAR≠
160  GOTO 70
170  PRINT A,B,C,≠ ≠,≠ ≠,≠ KOMPLEX≠
180  GOTO 70
190  PRINT ≠FALSCHE DATENEINGABE≠
200  GOTO 70
210  DATA -1, 6.2, 3, 1, 1, 1, 0, 0, 0
220  REM  SIEHE ABBILDUNG 11
900  END
```

98 ANSI Minimal BASIC, p. 24: "The parameter appearing in the parameter-list of a function definition is local to that definition; that is, it is distinct from any variable with the same name outside of the function definition." – Vgl. auch Garland, Dartmouth BASIC, p. 40.

99 ANSI Minimal BASIC, p. 39: "Defined functions may have zero or one parameter." Diese Erweiterung wird durch ANSI BASIC zum Standard erhoben.

100 ANSI Minimal BASIC, p. 25: "A function definition can refer to other defined functions, but not to the function being defined."

101 Vgl. auch S. 17 und S. 57.

Tabelle 10. Verfügbarkeit und Art des DEF-Statements

Prozessor	Einzeilige Anweisungsfunktion numerisch Anzahl der Formal-parameter	Einzeilige Anweisungsfunktion alphanumerisch Anzahl der Formal-parameter	Mehrzeilige Anweisungsfunktion numerisch	Mehrzeilige Anweisungsfunktion alphanumerisch	Formal-parameter Art
ANSI Minimal BASIC	1	n	n	n	einfache num. Variable
Dartmouth BASIC-6	bis zur maximalen Ausnutzung der Eingabezeile		v	v	einfache Variable
UOM BASIC/ UBASIC	bis zur max. Aus-nutzung d. Ein-gabezeile	n	v	n	einfache num. Variable
IBM CALL BASIC	1	n	n	n	einfache num. Variable
UWBIC	n	n	n	n	
FOSBIC (FOSBIC-Version 2)	n	n	n	n	(einfache num. Variable)

v = verfügbar
n = nicht verfügbar

Mehrzeilige benutzerdefinierte Anweisungsfunktionen – auch DEF-Blöcke genannt – sind nach ANSI BASIC zulässig.[101a] Für sie gilt

⟨Numerisches DEF-Statement⟩ : =
104 100

DEF FN ⟨Numerischer Funktions-identifikator⟩
102
Folge beliebiger BASIC-Statements ohne END- und ohne data-Statement, jedoch mit
LET FN ⟨Funktions-name⟩ = ⟨Arithmetischer Ausdruck⟩
62b 49
FNEND

Verschachtelte Definitionen sind unzulässig.[102] Ebenso darf nicht durch eine Programmsteuerungsanweisung[102a] von aussen zu einem Statement eines DEF-Blocks gesprungen werden. Programmverzweigungen innerhalb eines DEF-Blocks sind erlaubt.
Das folgende Programm besteht nur aus einer mehrzeiligen Anweisungsfunktion[103]; gezeigt wird auch, dass Funktionsaufrufe zur Schleifenverarbeitung führen können.

101a ANSI BASIC, p. 72–74.
102 Bull, Freeman and Garland, Specification for Standard BASIC, p. 75: "Function definitions may not be nested. That is, a definition block may not occur within another definition block. Function calls however may be nested in the usual way. Thus 100 PRINT FNM(0,FNM(1,2)) is acceptable for a suitably defined FNM." – Ähnlich auch Garland, Dartmouth BASIC, p. 41.
102a Vgl. Abs. 332.4.
103 Garland, Dartmouth BASIC, p. 40–41; Waite and Mather, BASIC, p. 64–65.

```
10  DEF FNM(X,Y)
20  LET FNM = M
30  IF Y <= X THEN 50
40  LET FNM = Y
50  FNEND
60  PRINT FNM(5,4),FNM(-5,-4)
70  PRINT FNM(1, FNM(2, FNM(3,0)))
80  END
```

Besonders zu beachten ist, dass zwischen DEF-Blöcken und der nach ANSI BASIC ebenfalls zugelassenen Unterprogrammtechnik mittels CALL-Statements[103a] insofern ein Unterschied besteht, als im DEF-Block alle Variablen ausser den in der Formalparameterliste erwähnten als sogenannte Globalvariable behandelt werden, deren Inhalt und Namen für das ganze Programm gelten. Eine Veränderung des Inhalts einer Globalvariablen in einem DEF-Block bewirkt, dass diese Inhaltsveränderung auch nach Abarbeitung des DEF-Blocks bestehen bleibt. In der Unterprogrammtechnik mittels CALL-Statement sind alle im Unterprogramm enthaltenen Variablen mit dem gleichen Namen wie im aufrufenden Programm unabhängig voneinander. Eine Inhaltsänderung einer Variablen im Unterprogramm wirkt sich nicht auf den Inhalt der Variablen mit dem gleichen Namen im Hauptprogramm aus.
Im folgenden Beispiel wird nach Aufruf der Funktion FNA in dem dazugehörigen DEF-Block der Inhalt der Globalvariablen K in Abhängigkeit von R auf 0 oder 1 gesetzt. Der bisherige Inhalt von K wird dabei zerstört.

```
100 DEF FNA(R)
110 IF R=1 GOTO 150
120 LET FNA=0
130 LET K=0
140 GOTO 170
150 LET K=1
160 LET FNA=1
170 FNEND
```

331.232 Alphanumerische DEF-Statements

Alphanumerische DEF-Statements können ebenfalls ein- oder mehrzeilig gestaltet werden; sie sind nur bei relativ wenigen BASIC-Compilern verfügbar.[104] Als Standard werden sie erst durch ANSI BASIC zugelassen.[105]
Für einzeilige Anweisungsfunktionen gilt

⟨Alphanumerisches DEF-Statement⟩ : =
105 100

103a Vgl. Bd. 2, Abs. 334.
104 Garland, Dartmouth BASIC, p. 39–41; Waite and Mather, BASIC, p. 65–66.
105 ANSI BASIC, p. 72–74.

DEF FN ⟨Alphanumerischer Funktionsidentifikator⟩ = ⟨Alphanumerischer Ausdruck⟩
106 50|51

⟨Alphanumerischer Funktionsidentifikator⟩ : = ⟨Buchstabe⟩ $ (⟨Alphanumerische Formalparameterliste⟩) |
106 105 3 107

⟨Buchstabe⟩ $
3

⟨Alphanumerische Formalparameterliste⟩ : = ⟨Einfache alphanumerische Variable⟩ | ⟨Alphanumerische Formalparameterliste⟩ , ⟨Einfache alphanumerische Variable⟩
107 106 45 107 45

Ein Vergleich der Formal- mit der Aktualparameterliste (Gleichung 66, Abs. 325.21) zeigt wiederum die strukturelle Gleichwertigkeit, bis auf den Unterschied, dass in der Formalparameterliste nur einfache alphanumerische Variable Verwendung finden dürfen.
Ein entsprechend strukturiertes DEF-Statement ist

```
10 DEF FNT$(A$,B$)=A$ & B$
```

mit dem jeweils die beiden Textinhalte von A$ und B$ zusammengesetzt werden.
Für mehrzeilige alphanumerische Anweisungsfunktionen gilt:

⟨Alphanumerisches DEF-Statement⟩ : =
108 100

DEF FN ⟨Alphanumerischer Funktionsidentifikator⟩
106
Folge beliebiger BASIC-Statements ohne END- und ohne data-Statement, jedoch mit
LET FN ⟨Funktionsname⟩ = ⟨Alphanumerischer Ausdruck⟩
62b 50|51
FNEND

Das nachstehende DEF-Blockbeispiel sucht in der alphanumerischen Variablen A$ alle Textteile, die dem Inhalt von B$ entsprechen und ersetzt sie durch den Textinhalt von C$.

```
 10 DEF FNR$(A$,B$,C$)
 20 LET N1=LEN(A$)
 30 LET N2=LEN(B$)
 40 IF N2>N1 GOTO 130
 50 FOR I=1 TO N1-N2+1
 60   IF B$<>A$(I,I+N2-1) GOTO 100
 70   X$=X$ & C$
 80   LET I=I+N2-1
 90   GOTO 110
100   X$=X$ & A$(I,I)
110 NEXT I
120 LET FNR$=X$
130 FNEND
```

331.233 Hybride DEF-Statements

Die hybriden DEF-Statements können in numerische und alphanumerische unterteilt werden.

⟨Hybrides DEF-Statement⟩ 108a 100 := ⟨Hybrides numerisches DEF-Statement⟩ 108b|108h | ⟨Hybrides alphanumerisches DEF-Statement⟩ 108c|108i

Durch diese DEF-Statements werden vom Benutzer Funktionen definiert, die einen Übergang zwischen numerischen und textlichen Inhalten bewirken. Dabei bestimmt das erzielte Ergebnis den – numerischen oder alphanumerischen – Grundcharakter des DEF-Statements.
Für das einzeilige hybride numerische DEF-Statement gilt:

⟨Hybrides numerisches DEF-Statement⟩ 108b 108a :=

DEF FN ⟨Hybrider numerischer DEF-Funktions-identifikator⟩ 108d = ⟨Arithmetischer Ausdruck⟩ 49

Der Funktionsidentifikator setzt sich aus einem Funktionsnamen und der Formalparameterliste zusammen.

⟨Hybrider numerischer DEF-Funktions-identifikator⟩ : = ⟨Funktions-name⟩ (⟨Hybride alphanumerische Formalparameterliste⟩)

108d 108b 62b 108e

Die hybride alphanumerische Formalparameterliste enthält mindestens eine alphanumerische Variable; die Liste kann zudem nur einfache Variable enthalten, während in der mit ihr korrespondierenden Aktualparameterliste[106] die den Variablen inhaltlich entsprechenden Ausdrücke erscheinen.

⟨Hybride alphanumerische Formalparameter-liste⟩ : = ⟨Einfache alphanumerische Variable⟩ |

108e 108d 45

⟨Einfache numerische Variable⟩ , ⟨Hybride alphanumerische Formalparameterliste⟩ |

37|38 108e

⟨Hybride alphanumerische Formalparameterliste⟩ , ⟨Hybride alphanumerische Formalparameterliste⟩

108e 108e

Ein Beispiel für ein einfaches hybrides numerisches DEF-Statement wäre demnach die Anweisung

```
10 DEF FNR(A$) = LEN(A$)
```

Durch sie wird die aktuelle Textlänge von A$ ermittelt.
Für das einzeilige hybride alphanumerische DEF-Statement gilt Entspechend:

⟨Hybrides alphanumerisches DEF-Statement⟩ : =

108c 108a

DEF FN ⟨Hybrider alphanumerischer DEF-Funktions-identifikator⟩ = ⟨Alphanumerischer Ausdruck⟩
108f 50|51

106 Vgl. Abs. 325.311, Gleichung 68g.

Der Funktionsidentifikator ist wie folgt strukturiert:

⟨Hybrider alphanumerischer DEF-Funktions-identifikator⟩ : = ⟨Funktions-name⟩ $ (⟨Hybride numerische Formalparameterliste⟩)

108f 108c 62b 108g

Die hybride numerische Formalparameterliste enthält mindestens eine einfache numerische Variable.

⟨Hybride numerische Formalparameter-liste⟩ : = ⟨Einfache numerische Variable⟩ |

108g 108f 37|38

⟨Einfache alphanumerische Variable⟩ , ⟨Hybride numerische Formalparameterliste⟩ |

45 108g

⟨Hybride numerische Formalparameterliste⟩ , ⟨Hybride numerische Formalparameterliste⟩

108g 108g

Durch die DEF-Anweisung

```
10 DEF FNA$(B$,N) = B$(1:N)
```

werden die ersten N Zeichen aus dem String B$ übertragen.
Hybride DEF-Statements lassen sich auch in mehrzeiligen DEF-Blöcken darstellen, wobei dem Grundcharakter des DEF-Statements entsprechend wiederum zwischen zwei verschiedenen Statementtypen differenziert werden kann.

⟨Hybrides numerisches DEF-Statement⟩ : =

108h 108a

DEF FN	⟨Hybrider numerischer DEF-Funktionsidentifikator⟩ 108d
	Folge beliebiger BASIC-Statements, ohne END- und ohne data-Statement, jedoch mit mindestens einem
LET FN	⟨Funktionsname⟩ = ⟨Arithmetischer Ausdruck⟩ 62b 49
FNEND	

⟨Hybrides alphanumerisches DEF-Statement⟩ : =
108i 108a

DEF FN	⟨Hybrider alphanumerischer DEF-Funktionsidentifikator⟩ 108f
	Folge beliebiger BASIC-Statements, ohne END- und ohne data-Statement, jedoch mit mindestens einem
LET FN	⟨Funktionsname⟩ $ = ⟨Alphanumerischer Ausdruck⟩ 62b 50|51
FNEND	

Durch das DEF-Blockbeispiel

```
 10 DEF FNX(A$,B$)
 20 LET N1=LEN(A$)
 30 LET N2=LEN(B$)
 40 IF N2>N1 GOTO 100
 50 FOR I=1 TO N1-N2+1
 60   IF B$<>A$(I,I+N2-1) GOTO 90
 70   LET FNX=I
 80   GOTO 110
 90 NEXT I
100 LET FNX=0
110 FNEND
```

wird die Anfangsposition des String B$ im String A$ ermittelt. Dies entspricht der Wirkungsweise der eingebauten Funktion POS(A$, B$); vgl. Tabelle 4e.

332 Ausführbare Anweisungen

Ausführbare Anweisungen unterscheiden sich von den Vereinbarungen und Kommentaren darin, dass durch sie die eigentliche Datenverarbeitung durchgeführt wird.
Wie Abbildung 19 zeigt, gliedern sich die ausführbaren Anweisungen in

Eingabe-, Wertzuordnungs-, Programmsteuerungs- und Ausgabeanweisungen.
Zunächst sollen die Eingabe- und Ausgabeanweisungen innerhalb der Programmiersprache BASIC erläutert werden. Es handelt sich um die aus Abbildung 22 ersichtlichen BASIC-Statements.

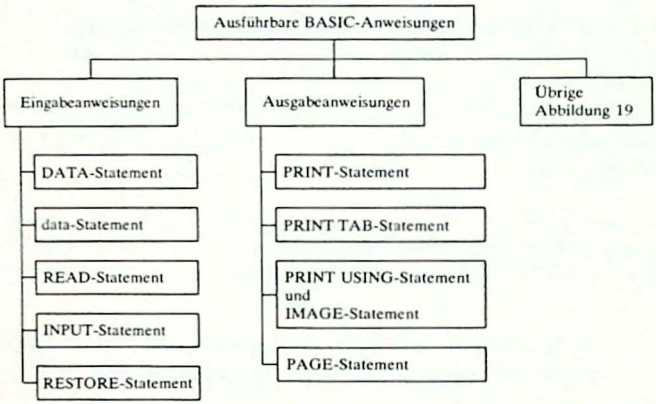


Abbildung 22.

332.1 Eingabeanweisungen

Die Eingabeanweisungen stellen Daten aus dem Eingabebereich in einer Pufferdatei bereit (DATA-Statement), übernehmen Daten aus der Pufferdatei in den Arbeitsbereich (Kernspeicher) des Computers (READ-Statement), wodurch die vorgesehenen Speicherbereiche einen Wertinhalt erhalten, und verwalten die Daten der Pufferdatei (RESTORE-Statement).
Die in der metalinguistischen Gleichung

$$\underset{109\quad 78}{\langle \text{Eingabe-anweisung} \rangle} := \underset{110}{\langle \text{DATA-Statement} \rangle} \mid \underset{114}{\langle \text{data-Statement} \rangle} \mid \underset{115}{\langle \text{READ-Statement} \rangle} \mid$$

$$\underset{117}{\langle \text{INPUT-Statement} \rangle} \mid \underset{118|119|120|121}{\langle \text{RESTORE-Statement} \rangle}$$

aufgeführten Eingabeanweisungen sind – mit Ausnahme des impliziten

data-Statements – in ANSI Minimal BASIC enthalten[107] und auch effektiv bei den meisten BASIC-Compilern verfügbar; vgl. Tabelle 11.

332.11 DATA-Statement

Durch DATA-Statements werden die zur Verarbeitung durch READ- (und MAT READ-)Statements benötigten Daten bereitgestellt.[108] DATA-Statements haben allgemein die folgende Struktur:

⟨DATA-Statement⟩ : =
110 109

DATA ⟨Konstantenliste⟩
111

⟨Konstantenliste⟩ : = ⟨Data-Konstante⟩ | ⟨Konstantenliste⟩ , ⟨Data-Konstante⟩
111 110 112|113 111 112|113

⟨Data-Konstante⟩ : = ⟨Compilerexterne numerische Konstante⟩ | ⟨Alphanumerische Konstante⟩
112 111 20 32

Aufgrund dieser Definition sind die folgenden DATA-Statements syntaktisch richtig:

```
900 DATA 99.99, 0.000
910 DATA -100, +93.1, -13.145E+12
920 DATA #NULL#, # #, 3.14159, #ENDE#
```

Numerische Data-Konstante sind generell im I- oder F-Format – bei manchen Compilern auch im E-Format – zulässig. Bezüglich der alphanumerischen Konstanten sei vermerkt, dass nach ANSI Minimal BASIC auch der nicht in Stringklammern gesetzte ⟨Stringinhalt⟩ als Data-Konstante zulässig ist.[109] Diese Option besteht auch bei Dartmouth BASIC-6,[110] so dass diesfalls

107 ANSI Minimal BASIC, p. 21–23.

108 ANSI Minimal BASIC, p. 22: "The data-statement provides for the creation of a sequence of representations for data elements for use by the read-statement." Garland, Dartmouth BASIC, p. 36: "The data statement provides for the creation of blocks of numeric and string data for use by the read statement."

109 ANSI Minimal BASIC, p. 22: "Each datum is either a numeric-constant, a string-constant, or an unquoted-string."

110 Garland, Dartmouth BASIC, p. 36.

$$\underset{113}{\langle \text{Data-Konstante} \rangle} := \underset{111}{\langle \text{Stringinhalt} \rangle} \mid \underset{34}{\left\langle \begin{matrix}\text{Compilerexterne} \\ \text{numerische Konstante}\end{matrix} \right\rangle} \mid \underset{20}{\left\langle \begin{matrix}\text{Alphanumerische} \\ \text{Konstante}\end{matrix} \right\rangle}$$

$$32$$

gilt. Vermerkt sei, dass der Nullstring weiterhin nur in der Form ' ' darstellbar ist.[111] Mithin ist das DATA-Statement

```
930 DATA 3.14159, PI, #,#
```

syntaktisch richtig; dagegen weisen die Statements

```
940 DATA 1, , 0
950 DATA 1.25E+340, 12.35, #E+340 IST ZU GROSS#
960 DATA #HIER FEHLT EINE STRINGKLAMMER, 0.000
```

Fehler auf.

Generell können DATA-Statements an jeder beliebigen Stelle im BASIC-Programm erscheinen; vorzugsweise werden sie am Programmende zusammengefaßt.

Jedes DATA-Statement bewirkt die lückenlose Zufügung der in ihm enthaltenen DATA-Konstanten an eine sequentiell organisierte Datei, deren Maximalumfang maschinenbezogen festgelegt ist und aus der die Daten durch READ- und MAT READ-Statements[112] nach dem FIFO-Prinzip (first in first out) zur Bearbeitung abgerufen werden.[113]

Nach ANSI Minimal BASIC ist für numerische und alphanumerische Daten eine Einheitsdatei zu bilden;[114] bei einzelnen BASIC-Compilern werden dagegen für die einzelnen Konstantentypen jeweils spezielle Dateien angelegt.[115]

111 ANSI Minimal BASIC, p. 38: "There is no provision for null data items, that is, for two adjacent commas in a data-statement. If the null string is to be included as a datum, it must be enclosed in quotation marks."

112 Vgl. Bd. 2, Abs. 331.11.

113 ANSI Minimal BASIC, p. 22: "Data from the totality of data-statements in the program are collected together into a single data sequence. ...The read-statement causes variables in the variable-list to be assigned values, in order, from the data sequence."

114 ANSI Minimal BASIC, p. 22: "There is a single sequence of data items rather than separate sequences for string data and for numeric data."

115 Garland, Dartmouth BASIC, p. 36: "Data from the totality of data statements in the program are collected together into two data blocks, one for numeric data and one for string data."

Tabelle 11. Eingabeanweisungen

Prozessor	DATA-Statement[1] Konstantenliste	data-Statement[1] Konstantenliste	READ-Statement	INPUT-Statement	RESTORE-Statement[2]
ANSI Minimal BASIC	I/F/E/A/G	I/F/E/A/G	v	v	118
Dartmouth BASIC-6	I/F/E/A/AS/G	I/F/E/A/AS/G	v	v	RESET
UOM BASIC/ UBASIC	I/F/E/A/AS/G	I/F/E/A/AS/G	v	v	118
IBM CALL BASIC	I/F/E/A/G	I/F/E/A/G	v	v	118/119
UWBIC	I/F	I/F	nur num. Variable	n	n
FOSBIC	I/F/E/A/G	I/F/E/A/G	v	v	118/121

[1] I = I-Format; F = F-Format; E = E-Format; A = Stringkonstante; G = gemischte Angabe möglich; AS = Stringinhalt nach Definitionsgleichung 34, wobei Einschränkungen in den Grundsymbolen compilerabhängig zu beachten sind.

[2] Die Zahlenangaben beziehen sich auf die Nummern der Definitionsgleichungen der RESTORE-Statements.

v = verfügbar

n = nicht verfügbar.

Der FOSBIC-Compiler lässt die Verarbeitung numerischer und alphanumerischer Data-Konstanten zu. Sich beim Dateiaufbau ergebende Fehler werden möglichst genau nachgewiesen.
Kennwortfehler bei DATA:

```
          YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

DATA-Statement ohne Data-Konstante:

```
          THERE ARE NO DATA IN THIS DATA STATEMENT
SEE BASIC TEXTBOOK  A 332.11
```

Exponent einer numerischen Konstanten überschreitet den im FOSBIC-Compiler maximal festgelegten Wert:

```
          I CAN ONLY HANDLE NUMBERS BETWEEN 10** 99 AND 10**(- 99)
SEE BASIC TEXTBOOK  A 322.12
```

Stringklammern, die bei alphanumerischen Daten generell erforderlich sind, treten nicht paarweise auf:

```
          THE QUOTATION MARKS ABOVE ARE NOT PAIRED
SEE BASIC TEXTBOOK  A  322.2
```

Data-Konstante in unbekanntem Format oder alphanumerisches Zeichen in numerischer Konstanten:

```
          WRONG CHARACTER WITHIN DATA
SEE BASIC TEXTBOOK  A 332.11
```

Überlauf der mittels DATA-Statements gebildeten Gesamtdatei (die normalerweise 500 Konstante umfassen kann):

```
          YOU GAVE ME    0 MORE DATA IN DATA STATEMENTS THAN I CAN HANDLE
SEE BASIC TEXTBOOK  A 332.11
```

332.12 data-Statement

Bei einigen BASIC-Compilern ist die Eingabe von Daten nach dem letzten Programmbefehl, dem END-Statement, möglich.
Diese Option ist auch beim FOSBIC-Compiler verfügbar, wobei allerdings ausschliesslich mit

⟨data-Statement⟩ : =
114 109

⟨Konstantenliste⟩ 111

zu arbeiten ist.

Die Anzahl der aufgrund von data-Statements einzulesenden Daten unterliegt keiner Beschränkung, da diese Daten nicht in einer Datei gespeichert, sondern nur nach Bedarf gelesen werden.
Das folgende Beispiel zeigt, wie zusätzliche Daten FOSBIC-konform durch data-Statements bereitgestellt werden.

```
900  REM PROGRAMMENDE BEI 999, ANSCHLIESSEND FOLGEN #DATA#
999  END
1., 2., 5., 18., 1
300.12, 11.22, -2
```

Fehlerhafte Formate und alphanumerische Zeichen in numerischen Konstanten werden durch den Ausdruck

```
PROGRAM STOPPED -- WRONG CHARACTER IN DATA
        END
                ERROR CODE=  5
            ***** SEE BASIC TEXTBOOK =  A 332.11  *****
            ***** I WAS AT LINE NUMBER    0
```

moniert. Generell können dieselben Fehlermeldungen wie bei dem DATA-Statement auftreten.

332.13 READ-Statement

Das READ-Statement bewirkt für die in ihm aufgeführten Variablen eine Wertzuweisung aus der mittels DATA-Statements organisierten Datei[116] und – bei deren Ausschöpfung – gegebenenfalls aus den mittels data-Statements bereitgestellten Daten.
READ-Statements weisen generell folgende Struktur auf:

⟨READ-Statement⟩ : =
115 109

READ ⟨Variablenliste⟩
116

⟨Variablenliste⟩ : = ⟨Variable⟩ | ⟨Variablen-liste⟩ , ⟨Variable⟩
116 115 35 116 35

Die numerischen und – falls zulässig – auch die alphanumerischen Variablen müssen nach den in Abschnitt 323 angeführten Regeln bezeichnet werden.

116 Garland, Dartmouth BASIC, p. 35: "The read statement provides for the assignment of values to variables from a block of data created by data statements."

Wenn ein READ-Statement mehrere Variable aufweist, wird die Liste von links beginnend nach rechts abgearbeitet, wobei der einer Variablen zugewiesene Wert in der Folge wiederum als Index einer anderen – im gleichen READ-Statement erscheinenden – Variablen verwendet werden kann. Dabei ist zu beachten, dass indizierte Variable in der Reihenfolge ihres Auftretens in den READ-Statements bewertet werden.
Durch die Anweisungsfolge

```
READ A, B, C
DATA 3, 8, 9
```

erhält die einfache Variable A den Wert 3, B den Wert 8 und C den Wert 9 zugewiesen. Durch die nachfolgende Anweisung

```
READ E(A,B), E(A,C)
DATA 50, 60
```

wird dem Element $E_{3,8}$ der (möglicherweise über das zu verarbeitende READ-Statement implizit dimensionierten)[117] Matrix E der Wert 50 und dem Element $E_{3,9}$ der Wert 60 zugewiesen. Vermerkt sei, dass sich die obigen READ- und DATA-Statements auch zusammenfassen lassen und durch

```
READ A, B, C, E(A,B), E(A,C)
DATA 3,8,9,50,60
```

oder

```
READ A, B, E(A,B), C, E(A,C)
DATA 3,8,50,9,60
```

ersetzbar sind.
Falls ein Compiler die Verarbeitung numerischer und alphanumerischer Variabler zulässt, können diese in den READ-Statements in beliebiger Reihenfolge erscheinen. Mithin wäre beispielsweise auch das READ-Statement

```
READ A$, B1, C
```

zulässig.
Die Abarbeitung der – möglicherweise mehrere Variable umfassenden – READ-Statements erfolgt nur soweit, wie die am temporären Dateiende (Abbildung 23) erscheinende numerische oder alphanumerische Konstante einer entsprechend definierten Variablen des READ-Statements

117 Vgl. Abs. 323.12.

zugeordnet werden kann und solange noch Daten in der Datei vorhanden sind (oder zusätzlich über data-Statements bereitgestellt werden).[118] Der FOSBIC-Compiler verarbeitet alphanumerische Zeichenketten nur bis zu einer Länge von maximal fünf Zeichen je Variable. Überschreitet eine alphanumerische Konstante in der DATA-Anweisung die Länge von fünf Zeichen, so wird diese Konstante in Bereiche zu je fünf Zeichen zerlegt, wobei der letzte Bereich – wenn notwendig – mit Leerzeichen auf fünf Zeichen angefüllt wird. Vgl. hierzu das folgende Programm und Druckbild.

```
 5  PRINT #1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678#
10  READ A$, B$, C$
20  DATA #TESTBEISPIEL#
30  PRINT A$, B$, C$
40  PRINT A$; B$; C$
50  END

1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
TESTB           EISPI           EL
TESTBEISPIEL
```

Die FOSBIC-Version 2 verfügt über eine flexiblere Textverarbeitung, die es erlaubt, Teilstrings mit dem READ-Statement einzulesen und an eine festgelegte Position der alphanumerischen Variablen zu bringen.
Durch die Anweisung

```
10 DATA "ABCDEFGHIJKLMNOPQRST"
20 READ A$(4:9)
```

werden in die Variable A$ ab Position 4 die Zeichen ABCDEF übertragen; der restliche Inhalt von A$ bleibt erhalten.
Mit

```
20 READ A$
```

wird der gesamte Textinhalt des obigen DATA-Statements übernommen, unter der Voraussetzung, dass die maximale Textlänge von A$ ausreichend genug dimensioniert ist. Trifft dies nicht zu, so wird der Textteil von links beginnend bis zur vollständigen Auffüllung des Inhalts von A$ erfasst. Überschreitet A$ in seiner maximalen Textlänge den Umfang des zu lesenden Textes, so wird der gesamte Text übernommen und das Textende im Inhalt von A$ durch ein spezielles Textendezeichen markiert; der Restinhalt von A$ ist undefiniert.

118 Vgl. ANSI Minimal BASIC, p. 23: "The type of a datum in the data sequence must correspond to the type of the variable to which it is to be assigned; that is, numeric-variables require unquoted-strings that are numeric-constants as data and string-variables require quoted-strings or unquoted-strings as data. An unquoted-string that is a valid numeric representation may be assigned to either a string-variable or a numeric-variable by a read-statement."

Mit dem READ-Statement verbundene Fehler werden vom FOSBIC-Compiler möglichst klar ausgewiesen.
Kennwortfehler bei READ:

```
        YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

READ-Statement enthält nicht nur Variablennamen:

```
        DATA CAN ONLY BE READ INTO A VARIABLE
        THE PART I DO NOT LIKE IS
SEE BASIC TEXTBOOK  A 332.13
```

Sonstige Fehler in der Variablenliste:

```
        I DO NOT LIKE THE EXPRESSION SHOWN BELOW (REASONS FOLLOW)
SEE BASIC TEXTBOOK  A  324.1
```

Datenliste ausgeschöpft:

```
PROGRAM STOPPED -- END OF DATA
                   ERROR CODE=  8
            ***** SEE BASIC TEXTBOOK =  A 332.13  *****
            ***** I WAS AT LINE NUMBER    0
```

Andere BASIC-Compiler erweisen sich bezüglich des letzterwähnten Fehlers als ebenso benutzerfreundlich; so erfolgt die Datenfehlanzeige vielfach durch OUT OF DATA IN ⟨Statementnummer⟩.

332.14 INPUT-Statement

Mit dem INPUT-Statement wird beim Time-Sharing-Betrieb eine unmittelbare Dateneingabe über den Terminal verlangt; dafür entfällt der Aufbau einer Datei über DATA-Statements und deren spätere Benutzung.[119]
Einzelne BASIC-Compiler verfügen zusätzlich über spezielle Statements zur erleichterten Eingabe von Strings: so ist bei Dartmouth BASIC-6 das LINPUT-Statement zur zeilenweisen Eingabe von (möglicherweise auch Kommata mitumfassenden) Daten zulässig.[120]
Das INPUT-Statement weist, analog zum READ-Statement die Struktur

119 Zur Fileverarbeitung vgl. Bd. 2, Abs. 332.
120 Garland, Dartmouth BASIC, p. 32: "The line input, or linput, statement enables the verbatum entry of a data which may contain commas or other characters that have special interpretations when used in response to an input statement." – Vgl. auch Waite and Mather, BASIC, p. 59, 168.

⟨INPUT-Statement⟩ : =
117 109

INPUT ⟨Variablenliste⟩ 116

auf und dient zur Eingabe numerischer und alphanumerischer Daten.[121] Beim Time-Sharing-Betrieb wird im Zuge des Programmablaufs bei Erreichung eines INPUT-Statements jeweils ein Fragezeichen (?) über den Terminal ausgegeben und damit vom Programmbenutzer die Eingabe von – normaliter über unmittelbar vorausgehende PRINT-Statements genau umschriebenen – Daten verlangt.
Beispielsweise würde das Programm

```
10 PRINT ≠WERT VON A UND B≠
20 INPUT A, B
30 PRINT ≠QUADRATWURZEL A * B =≠,SQR(A*B)
40 END
```

den Text WERT VON A, B mit Fragezeichen in der nächstfolgenden Zeile ausgeben. Der Programmablauf wird so lange unterbrochen, bis vom Programmbenutzer die Werte von A und B über den Terminal eingegeben werden; anschließend wird das Resultat der programmierten Rechenoperation ausgegeben.
Bei umfangreichen Programmen empfiehlt es sich, die vom Programmbenutzer einzugebenden Daten im voraus – etwa in Form von Benutzeranweisungen[122] – genau zu spezifizieren; dadurch kann der Programmablauf im allgemeinen wesentlich beschleunigt werden.
Der FOSBIC-Compiler gestattet die Verwendung von INPUT-Statements ebenfalls. Die über INPUT-Statements angeforderten Daten sind unmittelbar hinter dem END-Statement – durch entsprechende data-Statements – einzugeben. Damit wird es möglich, ursprünglich für den Dialogbetrieb entwickelte BASIC-Programme ohne weitere Umgestaltung FOSBIC-konform im Monologbetrieb zu verwenden.
Vermerkt sei, dass die Verwendung des INPUT-Statements bei Mono-

121 ANSI Minimal BASIC, p. 21: "An input-statement provides for interaction with a running program by allowing variables to be assigned values that are supplied from a source external to the program. The input-statement enables the entry of mixed string and numeric data, with data items being separated by commas."

122 Vgl. etwa: Siemens. Siemens System 4004. DIBA Dialog-BASIC-Anwenderprogramme. Teil I–III. München 1971/72.

logbetrieb auch nach ANSI Minimal BASIC vorgesehen ist.[123] Die vom FOSBIC-Compiler realisierte Form des INPUT-Statements entspricht den Erfordernissen der Stapelverarbeitung im Batchbetrieb.[124]
Der bevorzugte Einsatz von BASIC im Dialogbetrieb gab zur Weiterentwicklung der INPUT-Statements Anlass. Dies in dem Sinne, dass innerhalb der Anweisung eine – nach ANSI BASIC durch PROMPT zu kennzeichnende – alphanumerische Konstante in Stringklammern anzugeben ist, die bei Ausführung des INPUT-Statements ohne Stringklammern auf dem Bildschirm erscheint und zur Umschreibung der gemäss Variablenliste einzugebenden Daten dient.[124a]

⟨INPUT-Statement⟩ : =
117a 109

INPUT ⟨PROMPT-Identifikation⟩ ⟨Variablenliste⟩
117b 116

⟨PROMPT-Identifikation⟩ : = PROMPT ⟨Alphanumerische Konstante⟩ :
117b 117a 32

Weiterhin ist nach ANSI BASIC das LINPUT-Statement vorgesehen.[124b] Es dient zur Aufnahme alphanumerischer Inhalte, wobei auch Hochkomma und alle sonstigen zulässigen Zeichen erfasst werden können. Die Eingabe wird durch ein spezielles maschinenabhängiges end-of-line-Zeichen abgeschlossen.
Das LINPUT-Statement kann ohne oder mit PROMT-Identifikation verwendet werden.

123 ANSI Minimal BASIC, p. 21–22: "An input-statement causes the variables in the variable-list to be assigned, in order, values from the input-reply. ...In batch mode, the input-reply is requested from the external source by an implementation-defined means. Execution of the program is suspended until a valid input-reply has been supplied. ...An input-reply is terminated by an end-of-input-reply in both batch and interactive modes."
124 Vgl. Bd. 2, Abs. 512.
124a ANSI BASIC, p. 84–87.
124b ANSI BASIC, p. 84.

⟨INPUT-Statement⟩ : =
117c 109

LINPUT ⟨Alphanumerische Variablenliste⟩ 117d | ⟨PROMPT-Identifikation⟩ 117b ⟨Alphanumerische Variablenliste⟩ 117d

⟨Alphanumerische Variablenliste⟩ : = ⟨Alphanumerische Variable⟩ \|
117d 117c 44

⟨Alphanumerische Variable⟩ , ⟨Alphanumerische Variablenliste⟩
44 117d

Eine weitere, nach ANSI BASIC nicht vorgesehene, Modifikation des INPUT-Statements kann durch den Einbezug der TAB-Funktion erfolgen.[124c] Durch diesen Zusatz kann der Benutzer auf eine beliebige Zeile und Spalte des Bildschirms tabulieren, bevor eine eventuelle PROMT-Anweisung ausgeführt und/oder Dateneingabe entsprechend der Variablenliste erwartet wird.

332.15 RESTORE-Statements

Das RESTORE-Statement bewirkt in seiner einfachsten Form, dass eine via Data-Statements aufgebaute Datei – ab dem nächstfolgenden READ-Statement – nochmals von Beginn an gelesen werden kann.[125] Der RESTORE-Befehl erfolgt – ANSI Minimal BASIC entsprechend – durch das

⟨RESTORE-Statement⟩ : =
118 109

RESTORE

124c Zur TAB-Funktion vgl. Abs. 332.22.

125 ANSI Minimal BASIC, p. 23: "The restore-statement resets the pointer for the data sequence to the beginning of the sequence, so that the next read-statement executed will read data from the beginning of the sequence once again."

Einzelne BASIC-Compiler lassen auch die Form

⟨RESTORE-Statement⟩ : =
119 109

RESTORE ⟨Beliebige Folge von Grundsymbolen⟩
1

zu.

Die Wirkungsweise des RESTORE-Statements wird in Abbildung 23 schematisch dargestellt und kann auch anhand des folgenden Beispiels gut verfolgt werden.

```
10  DATA 1,2,3,4
20  READ X, Y, Z
30  PRINT #DRUCKBILD-1#, X, Y, Z
40  RESTORE
50  READ A, B, C, D
60  PRINT #DRUCKBILD-2#, A, B, C, D
70  END

DRUCKBILD-1          1            2            3
DRUCKBILD-2          1            2            3            4
```

Vermerkt sei, dass die Zurücksetzung des temporären Dateiendes auf den ursprünglichen Dateianfang beliebig oft erfolgen kann. Weiterhin bleiben RESTORE-Statements in Programmen ohne DATA-Statements unwirksam; sie führen auch zu keiner Fehlermeldung. Ein unmittelbar auf ein RESTORE-Statement folgendes Statement gleichen Typs bleibt ebenfalls wirkungslos.
Einzelne BASIC-Compiler erlauben auch das Zurücksetzen der getrennt aufgebauten numerischen und alphanumerischen Dateien, indem an das RESTORE-Statement nach Gleichung 118 ein * (für numerische Daten) oder ein $ (für alphanumerische Daten) angefügt wird.[126]

126 Sperry Rand. UNIVAC 1100 Series. BASIC. New York 1972, p. 3.3. – Vgl. auch Abs. 122.1.

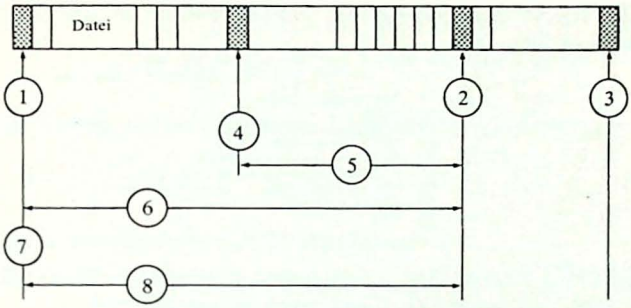


Abbildung 23. Dateiaufbau und -Benutzung durch DATA-, READ- und RESTORE-Statements

1 = Ursprünglicher Dateianfang – Erste Konstante des ersten DATA-Statements
2 = Temporäres Dateiende – Via DATA-Statements zuletzt eingefügte Konstante
3 = Potentielles Dateiende – Überlauf bei höherer Konstantenzahl
4 = Temporärer Dateianfang – Mit nächstfolgendem READ-Statement zuerst lesbare Konstante
5 = Temporärer Dateiumfang – Ohne RESTORE-Statement via READ-Statements noch lesbare Daten
6 = Gesamter Dateiumfang – Via DATA-Statements erstellte Datei
7 = Nach RESTORE-Statement mit nächstfolgendem READ-Statement zuerst lesbare Konstante
8 = Temporärer Dateiumfang nach RESTORE-Statement – Unmittelbar nach RESTORE-Statement via READ-Statements lesbare Daten

Als weitere Form des RESTORE-Statements sei

⟨RESTORE-Statement⟩ : =
120 109

RESTORE ⟨Statementnummer⟩
71

erwähnt. Durch diese Anweisung wird auf das erste Element der DATA-Anweisung mit der angeführten Statementnummer zurückgesprungen und dieses als neuer temporärer Dateianfang gesetzt.
Im folgenden Beispiel wird durch die Anweisung 60 das erste Element der Zeile 30 als neuer temporärer Dateianfang definiert. Mit dem darauf

folgenden READ-Statement erhält D(B) den Wert 10; der nächste lesbare Wert ist 20.

```
10  DIM D(10), C(8)
20  DATA 1,2
30  DATA 10, 20, 30, 40
40  READ A, B, C(A), C(B)
50  PRINT A, B, C(A), C(B)
60  RESTORE 30
70  READ D(B)
80  PRINT B, D(B)
90  END
```

Der FOSBIC-Compiler lässt – neben dem allgemeinen RESTORE-Statement nach Gleichung 118 – auch die Sonderform

⟨RESTORE-Statement⟩ : =
121 109

RESTORE ⟨Arithmetischer Ausdruck⟩ 49

zu. Dabei gibt der Wert des arithmetischen Ausdrucks an, um wie viele numerische und/oder alphanumerische Konstante der temporäre Dateianfang versetzt werden soll. Ein negativer Wert des arithmetischen Ausdrucks versetzt den temporären Dateianfang nach vorn, d. h. in Richtung auf das potentielle Dateiende; ein positiver Wert bewirkt die entgegengesetzte Versetzung (und damit eine entsprechende Ausweitung) des temporären Dateianfangs.
Angewendet auf das obige Beispiel ergibt sich folgende Programmodifikation:

```
10 DIM D(10),C(8)
20 DATA 1,2
30 DATA 10,20,30,40
40 READ A,B,C(A),C(B)
50 PRINT A,B,C(A),C(B)
60 RESTORE 2
70 READ D(B)
80 PPINT B,D(B)
90 END

1            2            10            20
2           10
```

Die Anwendungsmöglichkeiten des RESTORE-Statements sind vielfältig. Beispiel: Berechnung des durchschnittlichen Monatsumsatzes und der Quartalsumsätze bei einmaliger Eingabe der Grunddaten via DATA-Statements. Die letzterwähnte Sonderform des RESTORE-Statements lässt auch die Simulation einer Magnetbanddatei oder, bei entsprechendem Dateiaufbau, einer Magnetplattendatei zu.
Vom FOSBIC-Compiler werden Fehler im RESTORE-Statement differenziert moniert.

Kennwortfehler bei RESTORE:

```
          YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

Fehler im arithmetischen Ausdruck, numerische Konstante:

```
          ILLEGAL RESTORE STATEMENT
SEE BASIC TEXTBOOK  A 332.15
```

Fehler im arithmetischen Ausdruck, allgemein:

```
          I DO NOT LIKE THE EXPRESSION SHOWN BELOW (REASONS FOLLOW)
SEE BASIC TEXTBOOK  A  324.1
```

Unzulässige Verschiebung des Dateianfangs:

```
PROGRAM STOPPED -- YOU SET DATA POINTER ON POSITION NUMBER
-12 BUT I ONLY CAN DO IT UNTIL   330 OR ZERO
```

Überschreitung des potentiellen Dateiendes, Überlauf:

```
PROGRAM STOPPED -- YOU SET DATA POINTER ON POSITION NUMBER
512 BUT I ONLY CAN DO IT UNTIL   330 OR ZERO
```

Die beiden letzterwähnten Fehlermeldungen werden noch durch den Hinweis

```
      ERROR CODE=  7
***** SEE BASIC TEXTBOOK =  A 332.15  *****
***** I WAS AT LINE NUMBER   10
```

präzisiert.

332.2 Ausgabeanweisungen

Die Ausgabeanweisungen dienen zur Übertragung bestimmter Speicherplatzinhalte auf Datenträger, wobei die originären Ausgabedaten entweder nur zeitlich begrenzt (Sichtanzeige) oder unbegrenzt (Druckausgabe) verfügbar bleiben.
Für die BASIC-Ausgabeanweisungen gilt

⟨Ausgabe-anweisung⟩ := ⟨PRINT-Statement⟩ | ⟨PRINT TAB-Statement⟩ | ⟨PRINT USING-Statement⟩ |
122 78 123 126 129

⟨IMAGE-Statement⟩ | ⟨Allgemeines PRINT-Statement⟩ | ⟨PAGE-Statement⟩
132|145 150 152

mit der Einschränkung, dass nach ANSI Minimal BASIC nur die beiden ersterwähnten Statements vorgesehen sind. Nach ANSI BASIC liegen Standards für die ersten vier Statements vor.[127]

332.21 PRINT-Statement

Das PRINT-Statement weist generell die Struktur

⟨PRINT-Statement⟩ : =
123 122

PRINT ⟨Ausgabeliste⟩
124

auf. Es bewirkt bei seiner Ausführung die Generation einer an die Ausgabeeinheit zu übertragenden Zeichenkette, die nur aus Leerzeichen oder aus der sich bei sukzessiver Abarbeitung der Ausgabeliste ergebenden Zeichenfolge besteht.[128]
Wie sich aus der Bildungsregel

⟨Ausgabe-liste⟩ : = ⟨Arithmetischer Ausdruck⟩ | ⟨Alphanumerischer Ausdruck⟩ |
124 123 49 50|51

⟨Ausgabe-liste⟩ , ⟨Ausgabe-liste⟩ | ⟨Ausgabe-liste⟩ ; ⟨Ausgabe-liste⟩ |
124 124 124 124

⟨Abschluss-ausgabeliste⟩
125

ergibt, kann eine Ausgabeliste sowohl arithmetische als auch alphanumerische Ausdrücke umfassen. Einfachste arithmetische Ausdrücke sind numerische Konstante und numerische Variable. Anzahl und Reihenfolge der Ausdrücke in der Ausgabeliste können frei festgelegt werden. Nach vollständiger Abarbeitung der Ausgabeliste erfolgt Zeilenvor-

127 ANSI Minimal BASIC, p. 19–21; ANSI BASIC, p. 88–96.
128 ANSI Minimal BASIC, p. 19: "The execution of a print-statement generates a string of characters for transmission to an external device. This string of characters is determined by the successive evaluation of each print-item and print-separator in the print-list." – Vgl. auch Garland, Dartmouth BASIC, p. 29; Bull, Freeman and Garland, Specification for Standard BASIC, p. 92.

schub; innerhalb der Druckzeile geschieht die Datenausgabe nach Massgabe der verwendeten Drucksteuerzeichen (print-separators).
Nach ANSI Minimal BASIC sind Komma und Strichpunkt als Steuerzeichen vorgesehen. Das Komma bewirkt die Bereitstellung des nächstverfügbaren standardisierten Ausgabefeldes auf der bearbeiteten oder nächstfolgenden Zeile; der Strichpunkt führt zur Aufhebung der Standardeinteilung der Ausgabezeile zur gepackten Darstellung der Ausgabedaten.
Standardmässig wird die Ausgabezeile bei fast allen BASIC-Compilern in fünf gleich breite Felder eingeteilt, wobei Feldbreiten von 15 (Druck-) Stellen am häufigsten sind.[129]
Bezeichnet man mit d die Anzahl der signifikanten Stellen einer Zahl und mit e die Anzahl der Stellen eines möglichen Exponenten, so wird nach ANSI Minimal BASIC folgendes vorgeschlagen: Ganzzahlen zwischen $1 \leq x \leq 10^d$ werden mit allen Stellen ausgegeben. Die Stellenzahl beträgt mit Vorzeichen d + 2 (NR1 = implicit point unscaled notation). Gleitkommazahlen zwischen $0.1 - 0.5 \cdot 10^{(-d-1)} \leq x \leq 10^d - 0.5$ werden mit Dezimalpunkt und Anschlusszwischenraum ausgegeben. Zahlen kleiner 1 sollen keine Vorkommanull enthalten. Die Stellenzahl ist in diesem Fall mit Vorzeichen d + 3 (NR2 = explicit point unscaled notation). Alle anderen Zahlen ausserhalb der obigen Grenzen werden im E-Format mit e Stellen für den Exponenten ausgegeben. Die maximale Stellenzahl des E-Formats beträgt mit den beiden Vorzeichen, einem Dezimalpunkt, dem „E" und einem Anschlusszwischenraum d + e + 5 (NR3 = explicit point scaled notation). Die Zonenlänge soll d + e + 6 Zeichen betragen, unabhängig vom verwendeten Ausgabeformat, wobei d = 6 und e = 2 empfohlen werden
Literale erscheinen generell linksbündig ausgerichtet, wobei sich eine Zeichenkette auch über mehrere Felder erstrecken kann.

```
10  PRINT #1234567890123456789012345678901234567890123456789012345678901234567#
20  PRINT #TEXT#
30  PRINT #TEXTAUSGABE AB ERSTER STELLE#
40  END

1234567890123456789012345678901234567890123456789012345678901234567
TEXT
TEXTAUSGABE AB ERSTER STELLE
```

Zahlen werden in jedem Feld dezimalpunktgleich rechtsbündig ausgegeben. Die meisten Compiler lassen bei F-formatierten Werten maximal

129 ANSI Minimal BASIC, p. 20, 21: "Each print-line is divided into a fixed number of print zones, where the number of zones and the length of each zone is implementation defined. . . . A print line on a typical terminal might be devided into five print zones of fifteen print positions each."

sechs Nachkommastellen zu;[130] nachfolgenden Nullen werden in der Regel unterdrückt.
Vom FOSBIC-Compiler werden I- oder F-formatierte Werte – bis zur Vollausnützung eines Ausgabefeldes – auch rechtsverschoben dargestellt. Die in einem Datenfeld nicht in I- oder F-Format darstellbaren Zahlen werden automatisch in E-Format ausgegeben. Die für den FOSBIC-Compiler charakteristische Darstellungsweise ist aus dem folgenden Beispiel ersichtlich.

```
10  PRINT #123456789012345678901234567890123456789012345678901234567890123456789#
20  PRINT #9EI DIESEM PROGRAMM NUR DRUCKAUSGABE BEACHTEN#
30  LET F = 1
40  PRINT 0,F
50  FOR I = 1 TO 20
60    LET F = F*I
70    PRINT I, F, -1/F
80  NEXT I
90  END
```

Aus dem erzeugten Druckbild

```
123456789012345678901234567890123456789012345678901234567890123456789
BEI DIESEM PROGRAMM NUR DRUCKAUSGABE BEACHTEN
        0.0            1
        1              1              -1
        2              2               -.5
        3              6              -.166667
        4             24             -.0416667
        5            120            -.00833333
        6            720            -.00138889
        7           5040           -.000198413
        8          40320          -.0000248016
        9         362880         -.00000275573
       10        3628800        -2.75573E-7
       11       39916800        -2.50521E-8
       12      479002000        -2.08768E-9
       13     6227020000        -1.60590E-10
       14    87178300000        -1.14707E-11
       15  1307670000000        -7.64716E-13
       16    2.09228E+13        -4.77948E-14
       17    3.55687E+14        -2.81146E-15
       18    6.40237E+15        -1.56192E-16
       19    1.21645E+17        -8.22064E-18
       20    2.43290E+18        -4.11032E-19
```

ergibt sich, dass bei einer Feldbreite von 15 Stellen maximal 13 Stellen – bei Darstellung I-formatierter Zahlen – durch Ziffern belegbar sind; die zweite Stelle jedes Feldes ist ausschliesslich zur Ausgabe negativer Vorzeichen reserviert, und die erste Stelle ist nicht belegbar.[131]

130 Vgl. Garland, Dartmouth BASIC, p. 29–30.

131 ANSI Minimal BASIC, p. 19: "Numeric-expressions are evaluated to produce a string of characters consisting of a leading space if the number is positive or a leading minus sign if the number is negative followed by the decimal representation of the number and a trailing space."

Tabelle 12. Ausgabeanweisungen

Prozessor	Druck-zonen	Zeichen pro Druck-zone	PRINT-Statement maximal Zeichen pro Zeile	gepackte Ausgabe mit	Anzahl der signifikanten Ziffern in der Ausgabe als Ganzzahl	Gleitkommazahl	PRINT TAB-Statement	PAGE-Statement
ANSI Minimal BASIC		d + e + 6 (14)		;	6 = d	6 = d	v	n
Dartmouth BASIC-6	5	15	75	;	9	6	v	n
UOM BASIC/ UBASIC	5	4 × 15 + 1 × 12	72	;	8	8	v	n
IBM CALL BASIC	4	18	72	;	6	6	v	n
UWBIC	5	15	75	n	6	6	n	v
FOSBIC	1–8 einstellbar	15 oder einstellbar	120	;	6	6	v	v

v = verfügbar
n = nicht verfügbar
e = Zeichenzahl für Exponent

Prozessor	PRINT USING- und IMAGE-Statement Verfügbarkeit	Image-Zeichen	IMAGE-Kennung
ANSI Minimal BASIC	n	n	n
Dartmouth BASIC-6	v	# | − | + | · | $ | ^ | ⟨ | ⟩	
UOM BASIC/ UBASIC	PRINT IN FORM[1]		
IBM CALL BASIC	v	# | ! | & | CR | DB | B | · | à	:
UWBIC	n	n	n
FOSBIC	v	# | · | ↑	: | %

[1] Nur in einer UBASIC eigenen Form verfügbar. Vgl. Sperry Rand UNIVAC 1100 Series. BASIC. New York 1972, p. 3–30 bis 3–43.
v = verfügbar
n = nicht verfügbar

Fortsetzung von Tabelle 12. Ausgabeanweisungen

Bei F-formatierten Zahlen erscheint der Dezimalpunkt normaliter in der neunten Feldstelle; diese Stelle wird bei E-formatierten Zahlen durch E belegt.
Auf die links- bzw. rechtsbündige Datendarstellung ist bei der Spaltenbeschriftung – durch Textausrichtung innerhalb der über PRINT-Statements auszugehenden Strings – angemessen Rücksicht zu nehmen. Vgl. hierzu das folgende Beispiel.

```
10 PRINT ≠    ZEILEN-≠,≠   STUNDEN≠,≠       KOSTEN-≠
20 PRINT ≠    NUMMER ≠,≠          ≠,≠        SATZ≠
30 PRINT ≠ ≠,≠ ≠,≠         DM≠
40 PRINT 1,46,9.95
50 END

ZEILEN-          STUNDEN            KOSTEN-
NUMMER                               SATZ
                                      DM
     1              46                9.95
```

Das vorstehende Beispiel zeigt auch, wie mehrere PRINT-Statements zur mehrzeiligen Beschriftung von Ausgabefeldern verwendet und nicht zu

beschriftende Felder durch entsprechend in PRINT-Statements einbezogene leere Strings übersprungen werden können.
Der FOSBIC-Compiler gestattet es dem Benutzer, die Zahl der Ausgabefelder über eine dem BASIC-Programm vorangestellte *-Steuerkarte problemspezifisch festzulegen; normalerweise kann zwischen einem und acht Ausgabefeldern pro Zeile gewählt werden.[132] Bei Wegfall der Angabe in der *-Steuerkarte wird die Standardeinteilung von fünf Ausgabefeldern pro Zeile realisiert.
Flexible Ausgabefelddimensionierungen sind auch bei anderen BASIC-Compilern – teilweise direkt – möglich.[133] Das folgende Programm zeigt den Einsatz der FOSBIC-spezifischen Ausgabefeldsteuerung.

```
*3 DIE ERSTE ZAHL NACH DEM STERN
10 REM GIBT DIE DRUCKFELDERZAHL AN
20 READ A
30 PRINT A
40 IF A=100 GOTO 70
50 GOTO 20
60 DATA 1,2,3,4,5,6,7,8,9,10,11,12,100
70 END
*

    1          2          3
    4          5          6
    7          8          9
   10         11         12
  100
```

Für die meisten BASIC-Compiler ergeben sich aufgrund der Abschlussausgabeliste

$$\underset{125}{\langle}\text{Abschluss-}\underset{124}{\text{ausgabeliste}}\underset{125}{\rangle} := \langle\text{Leerzeichen}\rangle_{7} \mid \langle\text{Ausgabe-liste}\rangle_{124}, \mid \langle\text{Ausgabe-liste}\rangle_{124};$$

weitere Möglichkeiten zur Ausgabesteuerung.
Besteht die Ausgabeliste eines PRINT-Statements nur aus Leerzeilen, so erfolgt Vorschub zum ersten Ausgabefeld der nächsten Zeile. Das Statement PRINT ⟨Leerzeichen⟩ kann in unmittelbarer Folge beliebig oft wiederholt werden und führt diesfalls zur Ausgabe einer entsprechenden Zahl von Leerzeilen.[134] Die mit Komma oder Strichpunkt abgeschlossenen Ausgabelisten unterbinden dagegen den Zeilenvorschub.
Der Anwender sollte sich durch Testbeispiele intensiv mit den im PRINT-Statement verfügbaren Ausgabesteuerungsmöglichkeiten vertraut machen.

132 Vgl. Bd. 2, Abs. 533 und Tabelle 12.
133 Vgl. Kienzle. System 6100. Programmierhandbuch BASIC 6100. Villingen 1974, S. 122.
134 Bei Philips P350-S dient das LINE-Statement zur Generation von Leerzeilen. Vgl. Philips. BASIC 350-K Compiler für P350-S. Siegen 1974, S. 7.2.1.

Beim FOSBIC-Compiler ist – in Übereinstimmung mit ANSI Minimal BASIC[135] – gepackte Datenausgabe nur mittels Strichpunkt steuerbar; zwischen numerischen Werten wird ein Zwischenraum gelassen, während alphanumerische Werte gepackt als vollständig gekoppelte Zeichenkette ausgegeben werden.
Die FOSBIC-konformen Möglichkeiten zur Ausgabesteuerung sind aus dem folgenden Beispiel ersichtlich, wobei insbesondere auch die Auswirkung vorangehender Steuerzeichen auf das Statement
PRINT ⟨Leerzeichen⟩ zu beachten ist.[136] Vgl. auch Tabelle 13.

```
10 PRINT #123456789012345678901234567890123456789012345678901234567890123456789#
20 READ A, C$, B$, D$
30 PRINT A, A*A, A**A, EXP(A)
40 PRINT
50 PRINT A; A*A; A**A; EXP(A)
60 PRINT
70 PRINT C$; B$, A*A
80 PRINT C$; B$; A*A
90 PRINT D$; C$; B$, A*A, A**A, EXP(A)
100 PRINT D$; C$; B$; A*A; A**A; EXP(A)
110 DATA -3, #RESULTATE=#, #END#
120 END
```

```
12345678901234567890123456789012345678901234567890123456789012345678
      -3                9                -.037037       0.0497871

      -3 9 -.037037 0.0497871

RESULTATE=              9
RESULTATE= 9
ENDRESULTATE=           9                -.037037       0.0497871
ENDRESULTATE= 9 -.037037 0.0497871
```

Tabelle 13. Ausgabesteuerung mit PRINT-Statement.

(V/T/N)-Charakteristik	(T)-Wirkung bei Druckausgabe
(≡/≡/≡)	Zeilenvorschub, falls vorausgehendes PRINT-Statement mit (G/≡/≡) abgeschlossen; bei (G/,/≡) erfolgt dagegen Ausgabe von Leerzeichen in den verbleibenden Druckfeldern.
(G/≡/≡)	Zeilenvorschub.

135 ANSI Minimal BASIC, p. 19.
136 ANSI Minimal BASIC, p. 20: "A completely empty print-list will generate an end-of-print-line, thereby completing the current line of output. If this line containes no characters, then a blank line results."

(V/T/N)-Charakteristik	(T)-Wirkung bei Druckausgabe
(G/,/≡)	Übergang zum nächsten PRINT-Statement ohne Zeilenvorschub; Benützung des nächsten Ausgabefeldes nach den bei partiell abgearbeiteten Ausgabelisten geltenden Regeln.
(G/;/≡)	Übergang zum nächsten PRINT-Statement ohne Zeilenvorschub; Aufhebung der Standardeinteilung der Ausgabezeile und gepackte Druckausgabe nach den bei partiell abgearbeiteten Ausgabelisten geltenden Regeln.
(P/,/A)	Weiterbearbeitung des PRINT-Statements mit Druckausgabe im nächsten standardmässig freien Feld; vorausgehender Zeilenvorschub, falls fünftes Ausgabefeld bereits belegt.
(P/;/A)	Weiterbearbeitung des PRINT-Statements; Aufhebung der Standardeinteilung der Ausgabezeile und gepackte Druckausgabe, falls genügend Druckstellen verfügbar; andernfalls Zeilenvorschub und Druckausgabe im ersten Feld der neuen Zeile.
(P/,/α)	Weiterbearbeitung des PRINT-Statements mit Druckausgabe im nächsten standardmässig freien Feld; vorausgehender Zeilenvorschub, falls fünftes Ausgabefeld bereits belegt. Intermediärer Zeilenvorschub, falls der auszugebende alphanumerische Ausdruck das letzte Ausgabefeld voll ausnützt und Druckfortsetzung in der nächsten Zeile nötig wird.
(P/;/α)	Weiterbearbeitung des PRINT-Statements; Aufhebung der Standardeinteilung der Ausgabezeile und gepackte Druckausgabe; vorausgehender Zeilenvorschub, falls alle Druckstellen einer Zeile bereits ausgenützt sind. Intermediärer Zeilenvorschub, falls der auszugebende alphanumerische Ausdruck die Ausgabezeile voll ausnützt und Druckfortsetzung in der nächsten Zeile nötig wird.

Zeichenerklärung:

V = Abgearbeiteter Teil der Ausgabeliste, Gleichung 124.
- ≡ = Fehlende Ausgabeliste
- G = Gesamte Ausgabeliste
- P = Partielle Ausgabeliste

T = Zur Ausgabesteuerung verwendete Zeichen.
- ≡ = Leerzeichen
- , = Komma
- ; = Strichpunkt

N = Nachfolgendes Element des PRINT-Statements.
- ≡ = Leerzeichen
- A = Arithmetischer Ausdruck
- α = Alphanumerischer Ausdruck

Tabelle 13. Ausgabesteuerung mit PRINT-Statement, Fortsetzung

Die Möglichkeit zur Ausgabesteuerung bleibt auch voll erhalten, wenn ein PRINT-Statement primär Rechenanweisungsfunktionen übernimmt. So wird im folgenden Beispiel zwar in erster Linie der Barwert eines nach N Jahren fälligen Kapitals ermittelt, gleichzeitig aber auch bewirkt, dass die ersten zwei Druckfelder unbenützt bleiben und das Rechenergebnis – bei Textausgabe über zwei Druckfelder – im fünften Druckfeld erscheint.

```
10 PRIN: #123456789012345678901234567890123456789012345678901234567890123456789#
20 PRINT #ZINSESZINSRECHNUNG#
30 PRINT
40 READ S, I, N
50 PRINT # #,# #,#GEGENWARTSWERT = DM#,S*(1/((1+I)**N))
60 DATA 10000,10.5,3
70 END

123456789012345678901234567890123456789012345678901234567890123456789
ZINSESZINSRECHNUNG

                                GEGENWARTSWERT = DM                    6.57516
```

Vom FOSBIC-Compiler werden bei Verwendung des PRINT-Statements auftretende Fehler möglichst genau umschrieben.
Kennwortfehler bei PRINT:

```
          YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

Fehler bei alphanumerischem Ausdruck der Ausgabeliste:

```
          THE QUOTATION MARKS ABOVE ARE NOT PAIRED
SEE BASIC TEXTBOOK  A  322.2
```

Fehler bei arithmetischem Ausdruck der Ausgabeliste

```
          I DO NOT LIKE THE EXPRESSION SHOWN BELOW (REASONS FOLLOW)
SEE BASIC TEXTBOOK  A  324.1
```

mit Hinweis, dass
Ausdruck mit fehlender rechter Klammer:

```
B -- NO RIGHT PARENTHESIS FOR THIS LEFT PARENTHESIS
```

Ausdruck mit fehlender linker Klammer:

```
C -- NO LEFT PARENTHESIS FOR THIS RIGHT PARENTHESIS
```

Variable und Konstante unmittelbar nebeneinanderstehend:

```
L -- A VARIABLE IS NEXT TO A CONSTANT
```

Variable unmittelbar nebeneinanderstehend:

```
J -- THESE VARIABLES ARE NEXT TO EACH OTHER
```

Die beiden letzterwähnten Fehlermeldungen können insbesondere auch dann auftreten, wenn in einer Ausgabeliste zwei sich folgende arithmetische Ausdrücke nicht durch Komma oder Strichpunkt voneinander getrennt sind.

332.22 PRINT TAB-Statement

Das PRINT TAB-Statement ist eine Sonderform der PRINT-Anweisung, mit TAB-Funktion(en) in der Ausgabeliste.

⟨PRINT TAB-Statement⟩ : =
126 122

PRINT ⟨TAB-Ausgabeliste⟩ 127

⟨TAB-Ausgabeliste⟩ : = ⟨TAB-Funktion⟩ , ⟨Ausgabeliste⟩ | ⟨TAB-Funktion⟩ ; ⟨Ausgabeliste⟩ |
127 126 128 124 128 124

⟨Ausgabeliste⟩ , ⟨TAB-Ausgabeliste⟩ | ⟨Ausgabeliste⟩ ; ⟨TAB-Ausgabeliste⟩
124 127 124 127

⟨TAB-Funktion⟩ : = TAB (⟨Arithmetischer Ausdruck⟩) |
128 127 49

TAB (⟨Arithmetischer Ausdruck⟩ , ⟨Arithmetischer Ausdruck⟩)
49 49

Die TAB-Funktion erlaubt die beliebige Festlegung der bei Ausgabe einer Zeichenkette zu benutzenden Startposition und erleichtert damit insbesondere die Erstellung von Listen und graphischen Darstellungen.
Im Batchbetrieb und bei Ausgabe über den Schnelldrucker kann die TAB-Funktion nur mit einem arithmetischen Ausdruck verwendet werden, da eine eventuelle Zeilenrückpositionierung nicht möglich ist. Die Mehrzahl der BASIC-Compiler lässt jedoch auch ohne Fehlermeldung die TAB-Funktion mit Zeilenpositionierung auf dem Schnelldrucker zu, wobei allerdings diese Positionsangabe unterdrückt wird.
Nur im Dialogbetrieb und bei Ausgabe über den Terminalbildschirm ist die TAB-Funktion mit Spalten- und Zeilenpositionierung sinnvoll. Der erste arithmetische Ausdruck wird als Spalten- und der zweite als Zeilenposition interpretiert.

Nach ANSI BASIC ist in der TAB-Funktion nur ein arithmetischer Ausdruck zur Spaltenpositionierung zulässig.[136a]
Das folgende Beispiel zeigt die Wirkungsweise des PRINT TAB-Statements; es macht deutlich, dass das der TAB-Funktion folgende Trennzeichen (Komma oder Strichpunkt) unwirksam ist und die Ausgabe der TAB-gesteuerten Zeichenkette jedenfalls an der aufgrund der TAB-Funktion festgelegten Zeilenposition beginnt.

```
 10 PRINT #12345678901234567890123456789012345678901234567890123456789012345678#
 20 LET X = 0.10
 30 LET Y = 1000.0
 40 LET A$ = #AAAAA#
 50 LET B$ = #BBBBB#
 60 PRINT #A#, TAB(20), A$
 70 PRINT X, TAB(20), A$
 80 PRINT #B#; TAB(20); A$
 90 PRINT Y; TAB(20); A$
100 PRINT #C#, TAB(20); A$, TAB(40); B$
110 PRINT # #, TAB(20), A$, TAB(40), B$
120 PRINT 1000, TAB(20), A$, TAB(40), B$, X, Y
130 END

12345678901234567890123456789012345678901234567890123456789012345678
A                   AAAAA
        0.1         AAAAA
B                   AAAAA
     1000           AAAAA
C                   AAAAA               BBBBB
                    AAAAA               BBBBB
     1000           AAAAA               BBBBB          0.1          1000
```

Sinnvollerweise darf der Tabulationswert N nur zwischen 1 und der maximalen Spaltenzahl M einer Ausgabezeile liegen.
Beim FOSBIC-Compiler erfolgt – in Abweichung von ANSI Minimal BASIC[137] – Fehlermeldung, falls N < 0.

```
PROGRAM STOPPED -- NO NEGATIVE COLUMN-NUMBER IN PRINT TAB ALLOWED=****
                   ERROR CODE= 39
              ***** SEE BASIC TEXTBOOK =  A 332.22  *****
              ***** I WAS AT LINE NUMBER    0
```

Bei 0≤N<1.5 wird N auf 1 und in allen anderen Fällen auf die nächstliegende Ganzzahl gerundet. Ist N kleiner als die Zahl der in der Ausgabezeile bereits benutzten Druckpositionen, so erfolgt Zeilenvorschub mit nachfolgender Tabulierung auf N.[138] Andernfalls wird ohne Zeilenvorschub auf N tabuliert, solange N<M. Bei der irregulären Situation N>M

136a ANSI BASIC, p. 88.
137 ANSI Minimal BASIC, p. 20: "If n is less than one, an exception occurs."
138 ANSI Minimal BASIC, p. 37: "If TAB(N) is invoked when the current line already has more than N-1 characters, then printing resumes in column N on the next line. This convention does minimal damage to the structure of a table being generated by using tab-calls when some item in that table is too long to fit between successive tabular positions."

wird in der nächstfolgenden Zeile auf N modulo M tabuliert.[139] Vgl. hierzu das folgende Beispiel.

```
10 PRINT #1234567890123456789012345678901234567890123456789012345678901234567890#;
15 PRINT #1234567890123456789012345#
20 FOR I = 0 TO 20
30 PRINT TAB(8*I+1); I;
40 NEXT I
50 END

12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345
0.0     1       2       3       4       5       6       7       8       9
    10
            11
                    12
                            13
                                    14
                                            15
                                                    16
                                                            17
                                                                    18
19
        20
```

Bei anderen BASIC-Compilern erfolgt die Behandlung der PRINT TAB-Anweisung teilweise etwas anders; so werden etwa die Spalten einer Ausgabezeile von 0 beginnend numeriert[140] und bezüglich des Zeilenvorschubs andere Regeln eingehalten.[141]

332.23 PRINT USING- und IMAGE-Statement

Bei Verwendung von PRINT USING-Statements wird die Datenausgabe durch ein gekoppeltes IMAGE-Statement gesteuert und damit die Standardeinteilung der Ausgabezeile in normalerweise 5 × 15 Zeichen aufgehoben.
PRINT USING-Statements sind normalerweise wie folgt strukturiert:

⟨PRINT USING-Statement⟩ : =
129 122

PRINT ⟨Image-Ausgabeliste⟩
130

139 ANSI Minimal BASIC, p. 20. Vgl. auch Bull, Freeman and Garland, Specification for Standard BASIC, p. 94: "If the position is beyond the right margin, the value is taken modulo the value of the right margin."

140 Garland, Dartmouth BASIC, p. 30: "The columns are numbered from left to right starting with column zero." Bull, Freeman and Garland, Specification for Standard BASIC, p. 74: "The number of the leading column is 0." – Dies im Gegensatz zu ANSI Minimal BASIC, p. 37: "The columns in a print line are indexed starting with one rather than zero; that is, TAB(1) positions the typing mechanism in the first column of the line rather than in the second."

141 Vgl. Waite and Mather, BASIC, p. 50.

$$\left\langle \begin{matrix}\text{Image}\\ \text{Ausgabe-}\\ \text{liste}\end{matrix} \right\rangle_{130\ 129} := \text{USING} \left\langle \begin{matrix}\text{Image-}\\ \text{Statement-}\\ \text{kennung}\end{matrix} \right\rangle_{131} \Big| \text{USING} \left\langle \begin{matrix}\text{Image-}\\ \text{Statement-}\\ \text{kennung}\end{matrix} \right\rangle_{131},$$

$$\left\langle \begin{matrix}\text{Ausgabe-}\\ \text{liste}\end{matrix} \right\rangle_{124} \Big| \text{USING} \left\langle \begin{matrix}\text{Image-}\\ \text{Statement-}\\ \text{kennung}\end{matrix} \right\rangle_{131} : \left\langle \begin{matrix}\text{Ausgabe-}\\ \text{liste}\end{matrix} \right\rangle_{124}$$

$$\left\langle \begin{matrix}\text{Image-}\\ \text{Statement-}\\ \text{kennung}\end{matrix} \right\rangle_{131\ 130} := \langle \text{Arithmetischer Ausdruck} \rangle_{49} \Big| \left\langle \begin{matrix}\text{Alphanumerischer}\\ \text{Ausdruck}\end{matrix} \right\rangle_{50|51}$$

Die meisten BASIC-Compiler lassen zur Image-Statementerkennung nur einfach strukturierte arithmetische oder alphanumerische Ausdrücke zu; auch nach ANSI BASIC dürfen nur I-formatierte Ganzzahlen, alphanumerische Konstante oder alphanumerische Variable verwendet werden.[141a] Mithin gilt diesfalls für die Image-Statementkennung:

$$\left\langle \begin{matrix}\text{Image-}\\ \text{Statement-}\\ \text{kennung}\end{matrix} \right\rangle_{131\ 130} := \left\langle \begin{matrix}\text{I-formatierte}\\ \text{Ganzzahl}\end{matrix} \right\rangle_{22} \Big| \left\langle \begin{matrix}\text{Alphanumerische}\\ \text{Konstante}\end{matrix} \right\rangle_{32} \Big|$$

$$\left\langle \begin{matrix}\text{Alphanumerische}\\ \text{Variable}\end{matrix} \right\rangle_{44}$$

Hat die Image-Statementkennung einen numerischen Inhalt, so wird dieser als Bezugswert für eine Statementnummer genommen, die das entsprechende IMAGE-Statement enthält. Bei alphanumerischem Inhalt der Image-Statementkennung wird dieser Inhalt gleichzeitig als entsprechendes Image – vgl. Gleichung 134 – zur Aufbereitung des Inhalts der Ausgabeliste verwendet. Ein IMAGE-Statement ist in diesem Fall nicht notwendig; die alphanumerische Variable muss deshalb vor Ausführung des PRINT USING-Statements mit dem entsprechenden Image als Textinhalt gefüllt werden.[142]

141a ANSI BASIC, p. 92.

142 Sinnvollerweise geschieht dies durch ein alphanumerisches LET-Statement; vgl. Abs. 332.312.

Die Struktur des IMAGE-Statements zeigt die nachfolgende Definitionsgleichung:

⟨IMAGE-Statement⟩ : =
132 122

⟨Image-kennung⟩ 133	⟨Image⟩ 134

⟨Image-kennung⟩ : = IMAGE | : | %
133 132

Die meisten BASIC-Compiler verwenden als Imagekennung den Doppelpunkt (:) oder das Prozentzeichen (%). Nach ANSI BASIC wird auch das Kennwort IMAGE zugelassen.[142a]
Der FOSBIC-Compiler erlaubt die Verwendung beider Imagekennungen; die FOSBIC-Version 2 nimmt auch die Kennung durch IMAGE an.
Für die Image-Angaben des – im Programm an beliebiger Stelle einzufügenden – IMAGE-Statements gilt:

⟨Image⟩ : = ⟨Variablen-maske⟩ | ⟨String-inhalt⟩ ⟨Image⟩ |
134 132 135 34 134

⟨Image⟩ ⟨String-inhalt⟩ | ⟨Image⟩ ⟨Image⟩
134 34 134 134

⟨Variablen-maske⟩ : = ⟨Numerische Variablenmaske⟩ | ⟨Alphanumerische Variablenmaske⟩
135 134 136|144 143|149

⟨Numerische Variablen-maske⟩ : = ⟨I-Zeichen-kette⟩ | ⟨F-Zeichen-kette⟩ | ⟨E-Zeichen-kette⟩ |
136 135 137 138 139

⟨Monadischer Operator⟩ ⟨Numerische Variablenmaske⟩
11 136

142a Vgl. ANSI BASIC, p. 92.

⟨I-Zeichen-kette⟩ : = ⟨NR-Zeichen⟩ | ⟨NR-Zeichen⟩ ⟨I-Zeichen-kette⟩
137 136 140 140 137

⟨F-Zeichen-kette⟩ : = ⟨I-Zeichen-kette⟩ . ⟨I-Zeichen-kette⟩ | . ⟨I-Zeichen-kette⟩ |
138 136 137 137 137

⟨I-Zeichen-kette⟩ .
137

⟨E-Zeichen-kette⟩ : = ⟨I-Zeichen-kette⟩ ⟨EX-Zeichenkette⟩ |
139 136 137 141

⟨F-Zeichen-kette⟩ ⟨EX-Zeichenkette⟩
138 141

⟨NR-Zeichen⟩ : = # | §
140 137

Das §-Zeichen steht hier für compilerspezifische Nummerzeichen.

⟨EX-Zeichen-kette⟩ : = !!!! | ^^^^ | ⟨EX-Zeichen-kette⟩ ⟨EX-Zeichen⟩
141 139 141 142

⟨EX-Zeichen⟩ : = ! | ^
142 141

Die Bezeichnungen I-, F- und E-Zeichenkette beziehen sich auf die Darstellungsmöglichkeiten für Ganzzahlen und Gleitkommazahlen.[143]

⟨Alphanumerische Variablenmaske⟩ : = ⟨I-Zeichenkette⟩
143 135 137

Einzelne BASIC-Compiler lassen auch das $-Zeichen bei der Image-Angabe als führendes Zeichen zu, so dass

143 Vgl. Abs. 322.1.

$$\left\langle\begin{matrix}\text{Numerische}\\ \text{Variablen-}\\ \text{maske}\end{matrix}\right\rangle := \$\left\langle\begin{matrix}\text{Numerische}\\ \text{Variablen-}\\ \text{maske}\end{matrix}\right\rangle$$

144 135 136

gilt. Diese Option besteht beim FOSBIC-Compiler nicht; vgl. hierzu auch Tabelle 12.

Die folgenden Beispiele betreffen IMAGE-Statements mit numerischen Variablenmasken.

Der Zusammenhang zwischen PRINT USING- und IMAGE-Statement lässt sich anhand des Programms

```
10 PRINT #123456789012345678901234567890123456789#
20 PRINT USING 30, 1, 2.0, -3.45678
30         X     #          #.#          ##.#####
40 END

123456789012345678901234567890123456789
    1           2.0          -3.45678
```

leicht erkennen; gleichzeitig wird die mit dem IMAGE-Statement erreichbare grosse Flexibilität im Zeilendruck verdeutlicht.

Die bei Verwendung von PRINT USING-Statements – im Vergleich zu entsprechenden Programmen mit einfachen PRINT-Statements – erzielbare wesentliche Verbesserung der Tabellenausgabe wird mit dem folgenden Druckbild,

```
12345678901234567890123456789012345678901234567890123456789012345678
        OHNE PRINT USING                          MIT PRINT USING
      0.0            1        *                0    1.00000E+00
      1              1        *                1    1.00000E+00
      2              2        *                2    2.00000E+00
      3              6        *                3    6.00000E+00
      4             24        *                4    2.39999E+01
      5            120        *                5    1.19999E+02
      6            720        *                6    7.19999E+02
      7           5040        *                7    5.03999E+03
      8          40320        *                8    4.03199E+04
      9         362880        *                9    3.62879E+05
     10        3628800        *               10    3.62879E+06
     11       39916800        *               11    3.99167E+07
     12      479002000        *               12    4.79001E+08
     13     6227020000        *               13    6.22702E+09
     14    87178300000        *               14    8.71782E+10
     15  1307670000000        *               15    1.30767E+12
     16     2.09228E+13       *               16    2.09227E+13
     17     3.55687E+14       *               17    3.55687E+14
     18     6.40237E+15       *               18    6.40237E+15
     19     1.21645E+17       *               19    1.21645E+17
     20     2.43290E+18       *               20    2.43290E+18
```

das auf dem Programm

```
 10 PRINT #12345678901234567890123456789012345678901234567890123456789012345678#
 20 PRINT #          OHNE #,#PRINT USING#,# #,#MIT PRINT USING#
 30 %        ##     #.#####++++
 40 LET F = 1
 50 PRINT 0, F,TAB(32),#*#%
 60 PRINT USING 30, 0% F
 70 FOR I = 1 TO 20
 80   LET F = F * I
 90   PRINT I, F,TAB(32),#*#%
100   PRINT USING 30, I% F
110 NEXT I
120 END
```

basiert, verdeutlicht. Vermerkt sei, dass beim PRINT USING-Statement der Zeilenvorschub FOSBIC-konform durch Strichpunkt unterbunden wird.
Vom PRINT USING-Statement können – wie das folgende Beispiel zeigt – auch Rechenanweisungsfunktionen übernommen werden.

```
10 PRINT #123456789012345678901234567890123456789012345678901234567890#
20 LET A = 123.45
30 PRINT USING 40, A, A+1, A+2
40 %      ###.#
50 PRINT USING 60, A: A-0.45: A-23.45
60 %      ###.###  %
70 END

123456789012345678901234567890123456789012345678901234567890
     123.4
     124.4
     125.4
     123.449        123.000        99.999
```

Gemäss IMAGE-Statement nicht vorgesehene Dezimalstellen werden nicht ausgegeben; es erfolgt entsprechende Rundung. Nicht benutzte Dezimalstellen werden mit 0 aufgefüllt.
Das nachfolgende Beispiel umfasst IMAGE-Statements zur Datenausgabe im I-, F- und E-Format.

```
10 PRINT #123456789012345678901234567890123456789012345678901234567890#
20 %  I-FORMAT = ####   F-FORMAT = ###.##
30 %  E-FORMAT = #.#####^^^^
40 READ A, B, C
50 PRINT USING 20, A, B:
60 PRINT USING 30, C
70 DATA -123, 12.34, -1234.567
80 END

123456789012345678901234567890123456789012345678901234567890
  I-FORMAT = -123   F-FORMAT =  12.34  E-FORMAT = -.1234E+04
```

Beim FOSBIC-Compiler ist in der numerischen Variablenmaske der monadische Operator nicht speziell zu vermerken, da negative Zahlenwerte automatisch mit einem Minuszeichen versehen werden. Die für das Minuszeichen notwendige Stelle ist bei der Formatierung zu berücksichtigen. Vgl. hierzu das folgende Beispiel; nicht genügende Stellenzahl – inklusive Minuszeichen – wird durch Sterne im Druckbild angezeigt.

```
10 LET A = -123.456
20 PRINT USING 30, A, A
30 % #####.###       ####.##
40 PRINT USING 50, A, A
50 %  ####.#          ###.##
60 END

-123.456       -123.45
-123.4          ***.45
```

Eine Ausnahmeregelung besteht für die in F-Format auszugebenden Werte zwischen 0 und −1; bei ihnen wird – um eine Stelle für das Minuszeichen zu erhalten – der Dezimalpunkt um eine Position nach rechts verschoben und der Dezimalteil des Zahlenwerts entsprechend gekürzt.

```
5  PRINT #123456789012345678901234567 89#
10 LET B = -0.1234
20 PRINT USING 30, B, B, B
30         % ##.####  ##.###    .####
40 END

123456789012345678901234567 89
  -.1234   -.123    -.123
```

Bei mehrfach abzuarbeitenden IMAGE-Statements kann es sich als zweckmässig erweisen, den zwischen den Ausgabedaten einzuhaltenden Zwischenraum via IMAGE-Statement festzulegen; beim FOSBIC-Compiler erfolgt dies durch Einfügung des %-Zeichens in das IMAGE-Statement, wie das folgende Beispiel zeigt.

```
10 PRINT USING 20, 1, 2, 3, 4, 5, 6
20 %#%
30 PRINT USING 20, 1: 2: 3: 4: 5: 6
40 PRINT USING 50, 1: 2: 3: 4: 5: 6
50 %# %
60 END

 1
 2
 3
 4
 5
 6
 123456
 1 2 3 4 5 6
```

IMAGE-Statements mit I-Zeichenketten zur Ausgabe von Ganzzahlen bzw. zur Ausgabe von Text durch alphanumerische Variable enthält das folgende Programm

```
10 LET A$ = #ALPHA#
20 LET B$ = #BETA#
30 LET C$ = #GAMMA#
40 PRINT USING 50, 1234, A$, B$, C$
50 % #####
60 PRINT USING 50, 1234: A$: B$: C$
70 END

 1234
ALPHA
BETA
GAMMA
 1234 ALPHA BETA  GAMMA
```

Der FOSBIC-Compiler enthält folgende Fehlermeldungen zur syntaktischen Überprüfung des PRINT USING- und IMAGE-Statements:
Kennwortfehler bei PRINT, USING oder fehlendes Image-Statementzeichen:

```
        YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

Fehlende Statementnummer beim IMAGE-Statement:

```
        AN IMAGE STATEMENT MUST HAVE A STATEMENT NUMBER
SEE BASIC TEXTBOOK  A 332.23
```

Fehlende Image-Statementkennung im PRINT USING-Statement:

```
        THERE IS NO REFERENCE TO AN IMAGE STATEMENT WITHIN THE PRINT USING
SEE BASIC TEXTBOOK  A 332.23
```

Sobald die Anzahl der IMAGE-Statements eine einstellbare Grenze überschreitet, erscheint die Meldung:

```
        I CAN ONLY HANDLE  300 IMAGE STATEMENTS
SEE BASIC TEXTBOOK  A 332.23
```

Bei dem Versuch, mehr als eine einstellbare Anzahl von Zeichen pro Druckzeile auszugeben, wird das Programm mit dem Hinweis

```
PROGRAM STOPPED -- OUTPUT EXCEEDS  135 CHARACTERS AT IMAGE STATEMENT  300
                   ERROR CODE= 37
                ***** SEE BASIC TEXTBOOK =  A 332.23  *****
                ***** I WAS AT LINE NUMBER    0
```

abgebrochen.
Ebenfalls zu einem Programmabbruch führt eine fehlende IMAGE-Statementnummer.

```
PROGRAM STOPPED -- PERHAPS I AM CRAZY BUT I CANNOT FIND
                   AN IMAGE STATEMENT WITH THE NUMBER    12
                   ERROR CODE= 13
                ***** SEE BASIC TEXTBOOK =  A 332.23  *****
                ***** I WAS AT LINE NUMBER   10
```

Bei einzelnen Compilern weisen die IMAGE-Statements eine andere Struktur auf, indem die Image-Angaben in einer Variablen gespeichert werden, der sie mittels eines vorausgehend LET-Statements zugewiesen werden. In diesem Fall kann das IMAGE-Statement nicht an einer beliebigen Stelle im Programm stehen, sondern muss der entsprechenden PRINT USING-Anweisung logisch vorausgehen. Beispielsweise gilt dies auch für Dartmouth BASIC-6[144].
Ein im vorerwähnten Sinne verwendetes LET-Statement[144a] muss grundsätzlich so strukturiert sein, dass der alphanumerische Ausdruck (Konstante) das "image" enthält:

$$\text{LET} \quad \left\langle \begin{matrix}\text{Alphanumerische}\\ \text{Variable}\end{matrix} \right\rangle_{44} = \left\langle \begin{matrix}\text{Alphanumerischer}\\ \text{Ausdruck}\end{matrix} \right\rangle_{50|51}$$

Die Image-Angaben können sich ebenfalls um einen Zusatz – den Bündigkeitsregler – erweitern, so dass gilt:

$$\langle \text{Image} \rangle_{145\ 132} := \left\langle \begin{matrix}\text{Bündigkeits-}\\ \text{regler}\end{matrix} \right\rangle_{146} \Big| \left\langle \begin{matrix}\text{Bündigkeits-}\\ \text{regler}\end{matrix} \right\rangle_{146} \langle \text{Image} \rangle_{134}$$

144 Waite and Mather, BASIC, p. 50–57.
144a Zum LET-Statement vgl. Abs. 332.2.

$$\langle \text{Bündigkeits-regler} \rangle_{146,\,145} ::= < \mid > \mid \langle \text{Leerzeichen} \rangle_{7}$$

Das PRINT USING-Statement zu dem vorerwähnten IMAGE-Statement weist als Image-Statementkennung demnach eine einfache numerische oder alphanumerische Variable auf und unterscheidet sich dadurch wesentlich von der zuerst besprochenen Form der PRINT USING-Anweisung. Allerdings kann auch im PRINT USING-Statement selbst durch eine alphanumerische Konstante die Image-Angaben enthalten sein. Das folgende Beispiel zeigt den Einsatz der zweiten PRINT USING-Variante bei Verarbeitung numerischer Variablen.

```
10 LET Q$ = "I = ###.#   I*I =  ####.#"
20 FOR I=-1.5 TO 3.5 STEP 0.5
30   PRINT USING Q$: I,I*I
40 NEXT I
50 END
```

Vermerkt sei, dass der Bündigkeitsregler nur bei alphanumerischen Variablen zu verwenden ist; er bewirkt die links- bzw. rechtsbündige Ausgabe der Zeichenketten,[145] wobei nichtbenutzte Spalten frei bleiben und überlange Zeichenketten rechts- bzw. linksbündig abgeschnitten werden.[146]

Bei einzelnen BASIC-Compilern wird der Bündigkeitsregler in die Image-Angabe integriert, beispielsweise in der Form

$$\langle \text{Alphanumerische Variablenmaske} \rangle_{147,\,135} ::= \langle \alpha\text{-Zeichenkette} \rangle_{148}$$

$$\langle \alpha\text{-Zeichenkette} \rangle_{148,\,147} ::= \langle \alpha\text{-Zeichen} \rangle_{149} \mid \langle \alpha\text{-Zeichen} \rangle_{149} \langle \alpha\text{-Zeichenkette} \rangle_{148}$$

$$\langle \alpha\text{-Zeichen} \rangle_{149,\,148} ::= \mathrm{L} \mid \mathrm{R} \mid \mathrm{C} \mid \mathrm{E}$$

145 Waite and Mather, BASIC, p. 54–57.

146 Bull, Freeman and Garland, Specification for Standard BASIC, p. 97: "If the field began with a left justifier, then the places are filled starting with the leftmost reserved place and the first character of the string; surplus places are filled with spaces, and excess characters at the end of the string are ignored. If the field began with a right justifier, then the places are filled starting with the rightmost reserved place and the last character in the string; surplus places are filled with spaces, and excess characters at the beginning of the string are ignored."

Dabei kann über die α-Zeichenkette linksbündige (L), rechtsbündige (R) oder zentrierte (C) Schreibweise des Texts im entsprechend dimensionierten Feld gewählt werden. Grösste Flexibilität wird über die E-Zeichenkette erreicht; sie lässt die zur Vermeidung von Textabbrüchen notwendige automatische Ausweitung des Aufgabenfeldes zu.
Das folgende Programm bewirkt linksbündige Schreibweise nach den vorerwähnten Strukturierungsregeln.[147]

```
10  ! PREIS DM/100 KG == >LLLLLLLLLLLLLLLLLLLL #####.##
20  LET A$ = #HEIZOEL LEICHT      #
30  LET P = 35.40
40  PRINT USING 10, A$, P
50  END

PREIS DM/100 KG == HEIZOEL LEICHT                    35.40
```

Die Vielfalt des PRINT-USING- und IMAGE-Statements widerspricht dem ursprünglichen lernorientierten Charakter von BASIC. Die sich abzeichnende Tendenz in zunehmenden Mass Elemente von COBOL und FORTRAN in BASIC zu integrieren, beeinträchtigt die Geschlossenheit und Eigenständigkeit dieser Programmiersprache. Zukünftige Standardisierungsbestrebungen werden dadurch erheblich erschwert. Erst bei Einhaltung der ANSI BASIC Normen wird die Verwendung derartiger Statements auch für Austauschprogramme sinnvoll.[147a]

332.24 Allgemeines PRINT-Statement

Die neusten Entwicklungen auf dem Gebiet der Ausgabeanweisungen vereinen alle drei Formen des einfachen PRINT-Statements, des PRINT TAB-Statements und des PRINT USING-Statements, zu einem allgemeinen PRINT-Statement mit folgender Struktur[148]:

⟨Allgemeines PRINT-Statement⟩
150 122

PRINT ⟨Allgemeine Ausgabeliste⟩ 151

147 Eugene H. Barnett. Programming Time-Shared Computers in BASIC, New York 1972, p. 297–299.

147a Vgl. dazu ANSI BASIC, p. 92. – Es werden nur die Bündigkeitsregler nach Gleichung 146 genannt.

148 Ein derartiges allgemeines PRINT-Statement ist mit geringen Einschränkungen bei dem Computersystem Nixdorf 8870/1 und 3 mit Business-BASIC möglich; vgl. Nixdorf, System-Software Business-BASIC, S. 3.11, 3.17–3.21.

⟨Allgemeine Ausgabeliste⟩ : = ⟨Ausgabeliste⟩ | ⟨TAB-Ausgabeliste⟩ |
151 150 124 127

⟨Image-Ausgabeliste⟩ |
130

⟨Allgemeine Ausgabeliste⟩ , ⟨Allgemeine Ausgabeliste⟩ |
151 151

⟨Allgemeine Ausgabeliste⟩ ; ⟨Allgemeine Ausgabeliste⟩
151 151

332.25 PAGE-Statement

Das PAGE-Statement bewirkt, dass eine neue Seite des Ausgabeformulars zum Druck der (weiteren) Ergebnisse bereitgestellt wird.
Das Statement weist die einfache Form

⟨PAGE-Statement⟩ : =
152 122

PAGE

auf.

332.3 Wertzuordnungsanweisungen

Die Wertzuordnungsanweisungen stellen den dritten Typ der ausführbaren BASIC-Statements dar. Vgl. Abbildung 24.

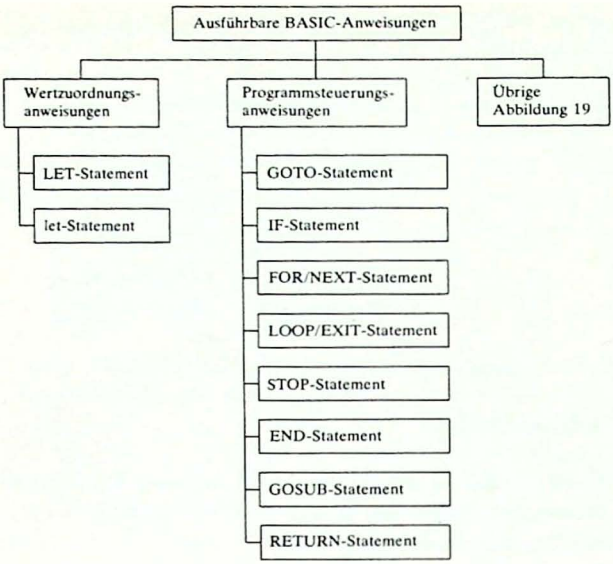


Abbildung 24.

Die Wertzuordnungsanweisungen werden in der amerikanischen BASIC-Literatur oft kurz als LET-Statements bezeichnet, was grundsätzlich richtig ist. Es gilt nämlich:

$$\underset{153}{\langle \text{Wertzuordnungs-anweisung} \rangle} \underset{78}{:=} \langle \text{LET-Statement} \rangle \underset{154|155|158}{|} \langle \text{let-Statement} \rangle \quad 161$$

Diese Statements bewirken generell die Zuordnung des – eventuell noch zu berechnenden – Werts eines BASIC-Ausdrucks an eine Variable.[149]

332.31 LET-Statements

In seiner einfachsten Form weist das LET-Statement die Struktur

149 ANSI Minimal BASIC, p. 16: "A let-statement provides for the assignment of the value of an expression to a variable." – Vgl. Abs. 324.

⟨LET-Statement⟩ : =
154 153

LET ⟨Variable⟩ = ⟨Ausdruck⟩
35 48

auf, wobei = als Zuordnungszeichen (assignment operator) zu interpretieren ist.
In der vorerwähnten Form ist das LET-Statement bei allen BASIC-Compilern verfügbar. Teilweise ist auch – vgl. Tabelle 14 – die Variante

⟨LET-Statement⟩ : =
155 153

LET ⟨Variablenliste⟩ = ⟨Ausdruck⟩
156|157 48

mit

⟨Variablenliste⟩ : = ⟨Variable⟩, ⟨Variable⟩ |
156 155 35 35

⟨Variablenliste⟩, ⟨Variable⟩
156 35

oder

⟨Variablenliste⟩ : = ⟨Variable⟩ = ⟨Variable⟩ |
157 155 35 35

⟨Variablenliste⟩ = ⟨Variable⟩
157 35

zulässig, wobei nach ANSI BASIC die ersterwähnte Form der Variablenliste als Standard vorgesehen ist.[149a] Mit dieser Statementform wird allen Variablen der Variablenliste der gleiche Wert zugewiesen.[150]

149a ANSI BASIC, p. 27: "The variable-list is a list of variables (of the same type) separated by commas."
150 Garland, Dartmouth BASIC, p. 24: "The value of the expression is computed and assigned to all variables in the let-list."

Grundsätzlich ist zwischen LET-Statements für numerische und alphanumerische Variable zu unterscheiden.

⟨LET-Statement⟩ : = ⟨Numerisches LET-Statement⟩ | ⟨Alphanumerisches LET-Statement⟩
158 153 159 160

332.311 Numerisches LET-Statement
Numerische LET-Statements vom Typ

⟨Numerisches LET-Statement⟩ : =
159 158

LET ⟨Numerische Variable⟩ = ⟨Arithmetischer Ausdruck⟩
36 49

bewirken, dass der Wert des rechts vom Zuordnungszeichen stehenden arithmetischen Ausdrucks der linksstehenden Variablen zugeordnet und auf deren Speicherplatz übertragen wird, wobei der seitherige Variablenwert verloren geht (destructive read-in).
Nach der unter 324.1 gegebenen Definition eines arithmetischen Ausdrucks ergibt sich, dass beispielsweise

```
LET P = 3.14159
```

ein korrekt strukturiertes LET-Statement darstellt. Mit ihm wird dem für P reservierten Speicherplatz der Wert von π, mit fünf Stellen nach dem Dezimalpunkt, zugewiesen und damit die sonst notwendige Eingabe über entsprechende READ-/DATA-Statements umgangen.
Besonders zu beachten ist, dass das LET-Statement keine Gleichung im mathematischen Sinn beinhaltet. Mithin ist es zulässig, beispielsweise durch

```
LET A = A + 1
```

den bisherigen Wert von A um 1.0 zu erhöhen und das sich ergebende Resultat neu dem Speicherplatz A zuzuweisen.
Vermerkt sei, dass ein LET-Statement auch zur impliziten Dimensionierung Anlass geben kann.[151] So wird etwa durch die Anweisung LET A(J) = 10/B implizit ein Vektor A dimensioniert; analog erfolgt durch LET R(K,4) = A(L)/B eine implizite Dimensionierung der Matrix R, sofern keine explizite Dimensionierung durch ein DIM-Statement vorausgegangen ist.

151 Vgl. Abs. 323.12.

Die besondere Bedeutung des LET-Statements liegt darin, dass mit ihm auch arithmetische Operationen programmiert und die bei ihrer Ausführung anfallenden Werte zur weiteren Verarbeitung bereitgestellt werden können. Vgl. hierzu das folgende Beispiel.

```
LET X1 = -B/2 + SQR(B**2/4 - A)
```

Bei der Bildung von LET-Statements ist die BASIC-Syntax genau einzuhalten. Beispielsweise sind die Statements LET A + B = C und LET −C = −B*A nicht BASIC-konform und sinnentsprechend nur in der Form

```
LET C = A + B
LET C = (-1)*(-B)*A
```

akzeptabel.
Bei Einhaltung der zur Bildung arithmetischer Ausdrücke geltenden allgemeinen Regeln können LET-Statements beliebig strukturiert werden. Vgl. hierzu das auf S. 97–98 angeführte Beispiel.
Das mehrfach numerische LET-Statement weist nach ANSI BASIC folgende Struktur auf:

⟨Numerisches LET-Statement⟩ : =
159a 158

LET ⟨Numerische Variablenliste⟩ = ⟨Arithmetischer Ausdruck⟩
159b 49

⟨Numerische Variablenliste⟩ : = ⟨Numerische Variable⟩ |
159b 159a 36

⟨Numerische Variablenliste⟩ , ⟨Numerische Variablenliste⟩
159b 159b

Diese Statementform kann auch von der FOSBIC-Version 2 verarbeitet werden, wobei die einzelnen Variablen allerdings durch ein Gleichheitszeichen (=) anstelle eines Kommas voneinander zu trennen sind. Entsprechendes gilt für eine Reihe weiterer BASIC-Compiler.[151a]

151a Vgl. Karl Weber. BASIC Programmier-Richtlinien für Austauschprogramme. Bern 1979, S. 74–77.

Bei Verwendung des FOSBIC-Compilers werden bezüglich des LET-Statements folgende Programmfehler ausgewiesen:
Kennwortfehler bei LET:

```
        YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

Mehr als eine oder keine Variable auf der linken Seite des LET-Statements:

```
        ITEM TO THE LEFT OF THE EQUAL SIGN MUST BE A VARIABLE
SEE BASIC TEXTBOOK  A  332.3
```

Kein Zuordnungszeichen:

```
        I CANNOT FIND AN EQUAL SIGN IN THE LET STATEMENT ABOVE
SEE BASIC TEXTBOOK  A  332.3
```

Falsche Stellung des Zuordnungszeichens:

```
        THE EQUAL SIGN ABOVE IS IN A FUNNY PLACE
SEE BASIC TEXTBOOK  A  332.3
```

Fehler im rechtsstehenden arithmetischen Ausdruck: vgl. Abs. 324.1.

332.312 Alphanumerisches LET-Statement
Leistungsfähige BASIC-Compiler lassen alphanumerische LET-Statements der Form

⟨Alphanumerisches LET-Statement⟩ : =
160 158

LET	⟨Variablen-liste⟩ 156|157	=	⟨Alphanumerischer Ausdruck⟩ 50|51

zu, wobei diese Form nach ANSI BASIC als Standard gilt.[151b]
Beispielsweise bewirkt das Programm

```
10 LET A$ = #BASIC#
20 PRINT A$
30 END
```

die Übernahme des Literals BASIC in den Speicherplatz A und den anschliessenden Druck dieses Literals.

151b ANSI BASIC, p. 27.

Mit Hilfe des LET-Statements können alphanumerische Variable auch umgespeichert und gekoppelt werden, was anhand des folgenden Beispiels dargestellt wird.

```
10  LET A$ = ≠BASIC≠
20  LET B$ = A$
30  LET A$ = ≠ANSI-≠
40  LET C$ = A$ + B$
50  PRINT ≠BENUTZTE SPRACHE ≠, C$
60  PRINT A$ + B$, A$: ≠MINIMAL ≠: B$
70  END

BENUTZTE SPRACHE                ANSI-BASIC
ANSI-BASIC      ANSI-MINIMAL BASIC
```

Die Verarbeitung von Stringlisten und -Tabellen ist nur bei besonders leistungsfähigen Compilern möglich. Beispielsweise lassen sich, entsprechende Dimensionierung vorausgesetzt, die Statements

```
110  LET T$ = A$(J+1)
120  LET A$(J+1) = A$(J)
```

mit Dartmouth BASIC-6 und dem FOSBIC-Compiler korrekt verarbeiten.
Analog zum numerischen LET-Statement wird von ANSI BASIC auch das mehrfache alphanumerische LET-Statement zugelassen.[151c]

⟨Alphanumerisches LET-Statement⟩ : =
160a 158

LET ⟨Alphanumerische Variablenliste⟩ = ⟨Alphanumerischer Ausdruck⟩
160b 50|51

⟨Alphanumerische Variablenliste⟩ : = ⟨Alphanumerische Variable⟩ |
160b 160a 44

⟨Alphanumerische Variablenliste⟩ , ⟨Alphanumerische Variablenliste⟩
160b 160b

Bei einzelnen BASIC-Compilern sind die in der Variablenliste enthaltenen alphanumerischen Variablen durch Gleichheitszeichen (=) statt Kommata voneinander zu trennen.
Durch die Teilzeichenkettenbegrenzung[151d] werden die Möglichkeiten des alphanumerischen LET-Statements erheblich erweitert, wie das nachfolgende Beispiel zeigt.

151c ANSI BASIC, p. 42.
151d Vgl. Abs. 323.23.

```
100 REM BEISPIEL Nixdorf 8870/1 TEILUNGSZEICHEN = ,
120 REM ALPHANUMERISCHE ADDITION MIT , ANSTATT & BEACHTEN !!!!
140 DIM T$(140),A$(140)
160 LET T$(1,49)="DURCH DIE TEILZEICHENKETTENBEGRENZUNG WERDEN DIE "
180 LET T$(50,105)="TEXTVERARBEITUNGSMöGLICHKEITEN MIT DEM ALPHANUMERISCHEN "
200 LET T$(106,140)="LET-STATEMENT erhöblich ERWEITERT!"
220 LET L9=80
240 LET A$=T$(1,L9-1)
260 PRINT A$
280 LET A$=T$(L9+1,140)
300 PRINT A$
320 LET A$=T$(120,138),T$(38,49),T$(50,119),T$(1,37),T$(139,139)
340 LET L9=66
360 PRINT A$(1,L9)
380 PRINT A$(L9+1,140)
400 END
```

```
DURCH DIE TEILZEICHENKETTENBEGRENZUNG WERDEN DIE TEXTVERARBEITUNGSMöGLICHKEITEN
MIT DEM ALPHANUMERISCHEN LET-STATEMENT erhöblich ERWEITERT!
erhöblich ERWEITERT WERDEN DIE TEXTVERARBEITUNGSMöGLICHKEITEN MIT
DEM ALPHANUMERISCHEN LET-STATEMENT DURCH DIE TEILZEICHENKETTENBEGRENZUNG!
```

Nach ANSI BASIC kann das alphanumerische LET-Statement auch benutzt werden, um alphanumerischen Variablen Image-Masken zuzuweisen, die in einem PRINT USING-Statement Verwendung finden.[151e] Dadurch entfallen die sonst üblichen IMAGE-Statements; ausserdem ergibt sich die Möglichkeit, in einem BASIC-Programm durch alphanumerische LET-Statements sukzessive verschiedene Image-Masken zusammenzustellen und so eine hoch flexible Ausgabegestaltung zu erreichen.
Das folgende Beispiel zeigt die Anwendung des alphanumerischen LET-Statements zur Image-Zuweisung.

```
10 DIM A$8, B$8
20 LET A$="########"
30 FOR I=1 TO 8
40   LET B$=A$(1:I)
50   PRINT USING B$:0.12345678*(10**I)
60 NEXT I
70 END

1
12
123
1234
12345
123456
1234567
12345678
```

Eine beachtenswerte Weiterentwicklung des alphanumerischen LET-Statements zum LET USING-Statement findet sich in Business-BASIC von Nixdorf.[151f] Diese Anweisung kombiniert LET-Statement und USING-Anweisungsteil des PRINT USING-Statements. Dadurch können numerische Inhalte entsprechend einer durch den USING-Teil angegebenen Image-Maske in alphanumerische Inhalte umgewandelt werden. Das LET USING-Statement kann mithin als hybrides alphanumerisches LET-Statement bezeichnet werden. Da es noch keiner Norm unterliegt und nur bei wenigen BASIC-Compilern verfügbar ist, soll hier lediglich seine Struktur – ohne Einbindung in das bestehende Gleichungssystem – definiert werden.

⟨Hybrides alphanumerisches LET-Statement⟩ : =

LET ⟨Alphanumerische Variablenliste⟩ = ⟨Arithmetischer Ausdruck⟩ USING 160b 49 ⟨Alphanumerische Variable | Konstante mit Image⟩

151e ANSI BASIC, p. 92–94.
151f Nixdorf, System-Software Business-BASIC, S. 10.11.

Ersetzt man im obigen Beispiel die beiden Anweisungen 40 und 50 durch

```
40   LET B$=0.12345678*(10**I) USING A$(1,I)
50   PRINT B$
```

, so entsteht derselbe Ergebnisausdruck.

332.32 let-Statements

Die meisten BASIC-Compiler lassen auch implizite let-Statements zu, die durch den Wegfall des Kennwortes LET erzeugt werden.[152] Für sie gilt generell

⟨let-Statement⟩ : = ⟨Numerisches let-Statement⟩ | ⟨Alphanumerisches let-Statement⟩
161 153 162 163

mit

⟨Numerisches let-Statement⟩ : =
162 161

let ⟨Numerische Variable⟩ = ⟨Arithmetischer Ausdruck⟩
36 49

und

⟨Alphanumerisches let-Statement⟩ : =
163 161

let ⟨Alphanumerische Variable⟩ = ⟨Alphanumerischer Ausdruck⟩
44 50|51

Ausserdem sind mehrfache numerische und alphanumerische let-Statements üblich, wobei die Formvorschriften der entsprechenden expliziten LET-Statements vollumfänglich gelten.
Implizite let-Statements sind nach ANSI Minimal BASIC nicht zulässig[152a], nach ANSI BASIC aber verwendbar.[152b]

152 Vgl. Weber, BASIC Programmier-Richtlinien für Austauschprogramme, S. 74–77.

152a ANSI Minimal BASIC, p. 36: "The keyword LET is required in minimal BASIC in order to make the language easier to learn. Although programs could be parsed by a language processor even if this keyword where omitted, novices find it easier to learn BASIC if every statement has a keyword; furthermore, this keyword serves to remind the user that an assignment statement does not assert a fact, but prescribes an action."

152b ANSI BASIC, p. 27.

Der FOSBIC-Compiler verarbeitet einfache, die FOSBIC-Version 2 auch mehrfache let-Statements; vgl. Tabelle 14.

```
•
10  A = SIN(3.14159)
20  B = COS(1.57079)
30  PRINT A, B
40  END
•
```

Prozessor	LET-Statement numerisch	alphanumerisch	ohne Kennwort LET
ANSI Minimal BASIC	154	161	n
Dartmouth BASIC-6	155 mit 157	160 mit 157	n
UOM BASIC/ UBASIC	155 mit 157	160 mit 157	v
IBM CALL BASIC	155 mit 156	160 mit 156	v
UWBIC	159	n	n
FOSBIC	154	v	v

v = verfügbar
n = nicht verfügbar
Die Zahlenangaben beziehen sich auf die Nummern der Definitionsgleichungen.

Tabelle 14. Verfügbarkeit und Art des LET-Statements.

332.4 Programmsteuerungsanweisungen

Die Leistungsfähigkeit eines Compilers in der Verarbeitung komplexer Programme hängt weitgehend von den verfügbaren Programmsteuerungsanweisungen (control statements) ab. Diese lassen sich, wie die metalinguistische Gleichung

$$\underset{164 \quad 78}{\left\langle \begin{array}{l}\text{Programm-}\\ \text{steuerungs-}\\ \text{anweisung}\end{array}\right\rangle} := \underset{165}{\left\langle \begin{array}{l}\text{Programm-}\\ \text{verzweigungs-}\\ \text{anweisung}\end{array}\right\rangle} \;\Big|\; \underset{166}{\left\langle \begin{array}{l}\text{Programm-}\\ \text{abschluss-}\\ \text{anweisung}\end{array}\right\rangle} \;\Big|\; \underset{167}{\left\langle \begin{array}{l}\text{Programm-}\\ \text{koppelungs-}\\ \text{anweisung}\end{array}\right\rangle}$$

zeigt, generell in drei Kategorien einteilen.

Zu den Programmverzweigungsanweisungen gehören GOTO-, IF-, FOR/NEXT- und LOOP/EXIT-Statements.

⟨Programm-verzweigungs-anweisung⟩ : = ⟨GOTO-Statement⟩ | ⟨IF-Statement⟩ | ⟨FOR/NEXT-Statement⟩ |
165 164 168 172 176|177|182

⟨LOOP/EXIT-Statement⟩
183|184|185

Diese Statements unterbrechen generell oder unter bestimmten Bedingungen die normalerweise sequentiell ablaufende Programmbearbeitung und bewirken den Übergang zu einem nicht unmittelbar folgenden Statement.[153] Während GOTO- und IF-Statements Sprunganweisungen im eigentlichen Sinne darstellen, bewirken FOR/NEXT- und LOOP/EXIT-Statements die Bildung zyklischer Schleifen.
Zur Beendigung des Programms führen die Programmabschlussanweisungen.

⟨Programm-abschluss-anweisung⟩ : = ⟨STOP-Statement⟩ | ⟨END-Statement⟩
166 164 186|187 188|189|190

Die Programmkoppelungsanweisungen dienen zum Aufruf von Unterprogrammen mit automatischem Rücksprung, wobei GOSUB den Sprung in das Unterprogramm und RETURN den Rücksprung in das Ausgangsprogramm bewirken.

⟨Programm-koppelungs-anweisung⟩ : = ⟨GOSUB-Statement⟩ | ⟨RETURN-Statement⟩
167 164 191 197|198|198a

332.41 GOTO-Statements

Jedes syntaktisch richtig angegebene GOTO-Statement bewirkt, dass ein Programm generell oder unter bestimmten Bedingungen nicht mit der ihm unmittelbar folgenden Anweisung fortgesetzt, sondern dass zu einer

153 ANSI Minimal BASIC, p. 16: "[These] control statements allow for the interruption of the normal sequence of execution of statements by causing execution to continue at a specified line, rather than at the one with the next higher line number."

anderen – im GOTO-Statement spezifizierten – Anweisung gesprungen und von dort aus weiter gearbeitet wird.
Alle BASIC-Compiler lassen zwei Typen von GOTO-Statements zu.

⟨GOTO-Statement⟩ : = ⟨Unbedingtes GOTO-Statement⟩ | ⟨Bedingtes GOTO-Statement⟩
168 165 169 170|170a

332.411 Unbedingte GOTO-Statements
Im unbedingten (simple) GOTO-Statement

⟨Unbedingtes GOTO-Statement⟩ : =
169 168

GOTO ⟨Statementnummer⟩ 71
GO TO ⟨Statementnummer⟩ 71
THEN ⟨Statementnummer⟩ 71

erscheint lediglich eine Statementnummer. Sie bezeichnet ein der GOTO-Anweisung vorausgehendes oder nachfolgendes Statement als Sprungziel, von dem aus das Programm weiter abgearbeitet werden soll.[154]
Die beiden ersterwähnten Statementformen sind nach ANSI BASIC alternativ verwendbar[154a]; das Kennwort THEN anstelle von GOTO findet sich nur bei älteren BASIC-Compilern und liegt heute ausserhalb jeder Norm. Vom FOSBIC-Compiler wird auch die Zeichenfolge THEN als Kennwort für das unbedingte GOTO-Statement erkannt.
Als Sprungziele können beliebige Anweisungstypen angegeben werden. Da nicht notwendigerweise auf ausführbare Anweisungen übergegangen werden muss, können insbesondere auch REM-Statements als Sprungziele angesetzt werden; dies im Gegensatz zu den FOSBIC-spezifischen

154 Garland, Dartmouth BASIC, p. 26: "A goto statement specifies that the next statement to be executed is the one whose line number is equal to the integer in the goto statement."
154a ANSI BASIC, p. 63–64.

COMMENT-Statements, die bei der Programmkompilierung unberücksichtigt bleiben und deshalb auch nicht als Sprungziele vorgebbar sind.[155] Unbedingte GOTO-Statements führen möglicherweise zu Programmschleifen aus denen nur durch weitere Sprunganweisungen herauszugelangen ist. Das unbedingte GOTO kann deshalb sinnvollerweise nur in direkter und indirekter Verbindung mit anderen Sprunganweisungen verwendet werden.

155 Vgl. Abs. 113.

Tabelle 15. Verfügbarkeit und Art der Programmverzweigungsanweisungen

Prozessor	Unbedingtes GOTO-Statement	Bedingtes GOTO-Statement		IF-Statement		FOR/NEXT-Statement	
			ganzzahliger Teil des arithm. Ausdrucks		Vergleich[1]	Zahl der ineinandergeschachtelten Schleifen	Schrittweite
ANSI Minimal BASIC	v	v	Rundung	v	54	10	0 unzulässig
Dartmouth BASIC-6	v	v	INT ⟨arith. Ausdruck⟩	v	54	8	0 zulässig
UOM BASIC/ UBASIC	v	v	INT ⟨arith. Ausdruck⟩	v	53/54	32	
IBM CALL BASIC	v	v	Abschneiden	v	54	15	0 unzulässig
UWBIC	v	n		v	54	20	0 unzulässig
FOSBIC	v	v	INT ⟨arith. Ausdruck⟩	v	54	einstellbar (CD 3300 = 20)	0 unzulässig

v = verfügbar
n = nicht verfügbar
[1] Die Zahlenangaben beziehen sich auf die Nummern der Definitionsgleichungen. Zur Verfügbarkeit einzelner Statementformen vgl. Weber und Türschmann, BASIC 1, 1. Aufl., S. 170; Weber, BASIC Programmier-Richtlinien für Austauschprogramme, S. 74–81.

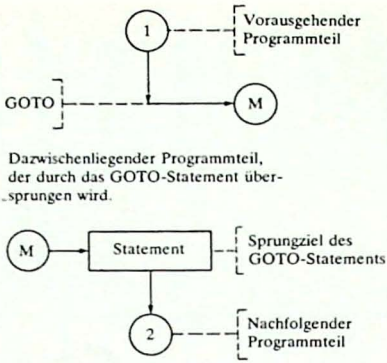


Abbildung 25. Flussdiagramm zum unbedingten GOTO-Statement

Dem Flowchart entsprechend wird der sequentielle Programmablauf mit dem unbedingten GOTO-Statement unterbrochen und mit dem Sprungzielstatement M fortgesetzt. Der übersprungene Programmteil muss, falls er benötigt wird, durch andere Programmverzweigungsanweisungen zur Ausführung gebracht werden.
Das folgende Programm umfasst lediglich ein unbedingtes GOTO-Statement als Sprunganweisung und führt zu einer endlosen Schleife (endless loop); ein Programmabbruch kann nur manuell (durch Eingriff der Operateure an der Konsole) oder maschinell (via vorgegebene Zeit- oder Druckzeilenlimits) erreicht werden.

```
*
    REMARK  PROGRAMM FUERHT ZU EINER ENDLOSEN SCHLEIFE
    REMARK  NICHT ALS TEXTPROGRAMM ZU VERWENDEN
    PRINT ≠SUMME VON GANZZAHLEN≠
    LET S = 0
    LET N = 1
    PRINT ≠ SUMME-1≠, ≠ ADDEND ≠, ≠ SUMME-2≠
10  PRINT S, N,
    LET S = S + N
    PRINT S
    LET N = N + 1
20  GOTO 10
30  END
*
```

Beim vorstehenden Beispiel könnte das Programmende gezielt dadurch erreicht werden, dass zwischen den Statements 10 und 20 ein IF-Statement – etwa der Art IF S GE 100 GOTO 30 – eingebaut würde. Zu den IF-Statements vgl. den folgenden Abschnitt.
Die kombinierte Anwendung von GOTO- und IF-Statements ist auch aus dem folgenden Programm – Berechnung des arithmetischen und quadratischen Mittels einer beliebigen Menge von Zahlen – ersichtlich.

```
10  LET N=0
20  LET S1=0
30  LET S2=0
40  READ A
50  IF A$=≠ENDE≠ GOTO 100
60  LET N=N+1
70  LET S1=S1+A
80  LET S2=S2+A**2
90  GOTO 40
100 PRINT ≠ANZAHL DER EINGELESENEN ZAHLEN =≠,N
110 PRINT ≠SUMME=≠,S1
120 PRINT ≠QUADRATSUMME=≠,S2
130 PRINT ≠ARITHMETISCHES MITTEL=≠,S1/N
140 PRINT ≠QUADRATISCHES MITTEL =≠,SQR(S2)
150 DATA 1,2,3,4,5,6,7,8,9,10,11,≠ENDE≠
160 END

ANZAHL DER EINGELESENEN ZAHLEN =
SUMME=                    66
QUADRATSUMME= .          506
ARITHMETISCHES MITTEL=                6
QUADRATISCHES MITTEL =               22.4944
```

Verweist ein unbedingtes GOTO-Statement auf die Statementnummer einer nicht ausführbaren Anweisung, so fährt das Programm mit der nächstfolgenden ausführbaren Anweisung fort.

```
10  REM SPRUNGZIELE
20  GOTO 120
30  DIM A(20,20)
40  PRINT ≠DIM ALS SPRUNGZIEL ANGENOMMEN≠
50  GOTO 150
120 DATA 20,30
130 PRINT ≠DATA ALS SPRUNGZIEL ANGENOMMEN≠
140 GOTO 30
150 END

DATA ALS SPRUNGZIEL ANGENOMMEN
DIM ALS SPRUNGZIEL ANGENOMMEN
```

332.412 Bedingte GOTO-Statements

Das bedingte (computed) GOTO enthält eine Statementnummernliste und einen arithmetischen Ausdruck. Bei Ausführung des GOTO-Statements wird aufgrund des Wertes des arithmetischen Ausdrucks eine in der Statementnummernliste enthaltene Statementnummer als Sprungziel verwendet (oder – allerdings nicht bei allen Compilern – mit dem nächstfolgenden Statement fortgefahren).

Bedingte (oder verzweigte) GOTO-Statements können – vgl. Tabelle 15 – folgende Formen aufweisen:

⟨Bedingtes GOTO-Statement⟩
170 168

ON	⟨Arithmetischer Ausdruck⟩ 49	GOTO	⟨Statement-nummernliste⟩ 171	|
ON	⟨Arithmetischer Ausdruck⟩ 49	GO TO	⟨Statement-nummernliste⟩ 171	|
OF	⟨Arithmetischer Ausdruck⟩ 49	GOTO	⟨Statement-nummernliste⟩ 171	|
GOTO	⟨Statement-nummernliste⟩ 171	ON	⟨Arithmetischer Ausdruck⟩ 49	|
GOTO	⟨Statement-nummernliste⟩ 171	OF	⟨Arithmetischer Ausdruck⟩ 49	|
ON	⟨Arithmetischer Ausdruck⟩ 49	THEN	⟨Statement-nummernliste⟩ 171	|
OF	⟨Arithmetischer Ausdruck⟩ 49	THEN	⟨Statement-nummernliste⟩ 171	|
THEN	⟨Statement-nummernliste⟩ 171	ON	⟨Arithmetischer Ausdruck⟩ 49	|
THEN	⟨Statement-nummernliste⟩ 171	OF	⟨Arithmetischer Ausdruck⟩ 49	

Nach ANSI Minimal BASIC und ANSI BASIC sind nur die beiden ersten Formen als Standard zugelassen.[156] Die Notwendigkeit der Standardisierung zeigt sich gerade hier an der Vielzahl der Formen des bedingten GOTO-Statements; nur durch Normung kann diese Dialektvielfalt unterbunden und eine hohe Transportfähigkeit der BASIC-Programme gewährleistet werden.
Die Statementnummernliste ist eine Folge einzelner, durch Kommatas getrennter, Statementnummern.

$$\underset{171}{\langle \text{Statement-nummern-liste} \rangle} \underset{170}{:=} \underset{71}{\langle \text{Statement-nummer} \rangle}, \underset{71}{\langle \text{Statement-nummer} \rangle} \;\Big|\; \underset{170}{\langle \text{Statement-nummern-liste} \rangle}, \underset{171}{\langle \text{Statement-nummern-liste} \rangle}$$

Der Wert des in einem GOTO-Statement erscheinenden arithmetischen Ausdrucks wird bei jedem Programmdurchlauf neu ermittelt und nötigenfalls durch Rundung (rounding)[157] oder Abschneiden des Dezimalteils (truncation)[158] ganzzahlig gemacht; vgl. Tabelle 15. Mit dem ganzzahligen Wert wird normalerweise aus der Statementnummernliste eine Statementnummer als Sprungziel aktualisiert, wobei zuvor compilerintern diese Liste von links mit 1 beginnend durchnumeriert wurde. Falls der ganzzahlige Wert des arithmetischen Ausdrucks kleiner als 1 oder grösser als die Zahl der angegebenen Statementnummern ist, wird mit der nächstfolgenden Anweisung fortgefahren oder eine compilerspezifische Fehlermeldung angegeben.[159]

156 ANSI Minimal BASIC, p. 16–17; ANSI BASIC, p. 63–65. – Das Kennwort THEN anstelle von GOTO wird bei älteren BASIC-Compilern verwendet; vgl. Waite and Mather, BASIC, p. 169; Sperry Rand UNIVAC 1100 Series. BASIC. New York 1972, p. 3.8.

157 ANSI Minimal BASIC, p. 36: "The value of the expression in the on-goto-statements is to be rounded, rather then truncated, in order to be consistent with the treatment of subscript expressions for arrays."

158 Garland, Dartmouth BASIC, p. 27: "The expression in an on statement is evaluated and truncated to obtain a positive integer, whose value is then used to select an integer from the list following the GOTO."

159 Nach ANSI Minimal BASIC wird seit dem 30. 7. 75 die zweite Lösung bevorzugt: "If the value of the expression in the on-goto-statement cannot be used to select a line-number from the list, an error conditions is to results. (Voted 75-07-30, 15 to 11, reversing a vote on 74-10-22 of 14 to 12 in favour of having execution continue with the next line; position reaffirmed 75-10-29, 21 to 5.)" – ANSI Minimal BASIC, p. 36.

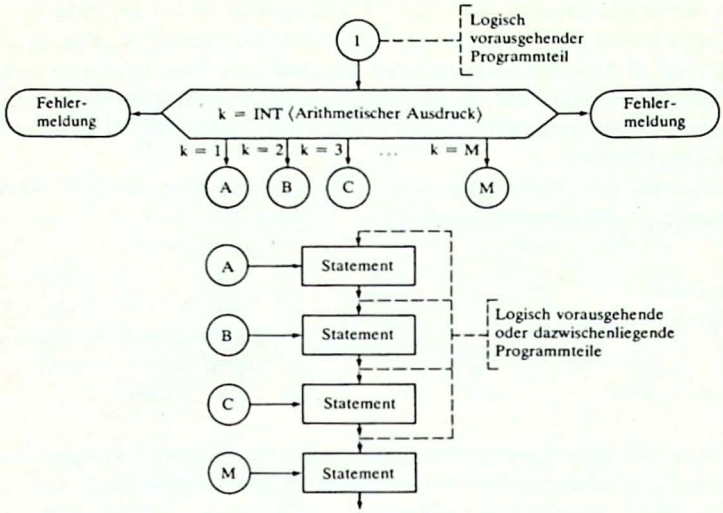


Abbildung 26. Flussdiagramm zum bedingten GOTO-Statement

Zum Beispiel erfolgt bei dem Statement

```
100  GOTO 400, 500, 999 ON (X+3)/3
```

dann ein Übergang zu 999, wenn X = 7 ist, da dann (7 + 3)/3 = 3.333 auf 3 gesetzt wird.

Beachtenswert ist, dass sich mehrere bedingte GOTO-Statements unmittelbar und aufeinander abgestimmt folgen können, falls ein GOTO die Liste der Sprungziele nicht vollumfänglich aufzunehmen vermag. Diesfalls kann, wie das Beispiel

```
  1 REM N = KLEINSTER WERT VON I
  2 REM M = GROESSTER WERT VON I
  3 REM K = ANZAHL DER STATEMENTNUMMERN JE BEDINGTEN GOTO-STATEMENT
  4 IF I > M GOTO 999
  5 IF I < N GOTO 999
  6 LET I = I-N+1
  7 GOTO 10, 20, 30, . . . . . ON (I-1)/K+1
 10 GOTO 100, 110, 120, . . . .  ON I
 20 GOTO 200, 210, 220, . . . .  ON I-K
 30 GOTO 300, 310, 320, . . . .  ON I-2*K
 40 GOTO 400, 410, 420, . . . .  ON I-3*K
    ....
    ....
999 PRINT "SPRUNGFEHLER"
    ....
    ....
```

zeigt, eine sukzessive Verbindung der einzelnen GOTO-Statements dadurch erreicht werden, dass der Wert des zur Sprungzielbestimmung ver-

wendeten arithmetischen Ausdrucks beim Übergang zum nächstfolgenden GOTO-Statement um die Zahl der in der vorhergehenden Statementnummernliste enthaltenen Elemente reduziert wird.
Der FOSBIC-Compiler lässt alle Formen von bedingten GOTO-Statements zu, was insbesondere auch die unveränderte Übernahme von BASIC-Austauschprogrammen erleichtert.
Nach ANSI BASIC ist auch die folgende Form des bedingten GOTO-Statements zulässig[159a]:

⟨Bedingtes GOTO-Statement⟩ : =
170a 168

ON ⟨Arithmetischer Ausdruck⟩ GOTO ⟨Statement-nummern-liste⟩ ELSE ⟨Imperatives Statement⟩
49 171 175

Durch diese Statementform kann die Fehlermöglichkeit abgefangen werden, dass der arithmetische Ausdruck einen ganzzahligen Wert hat, der kleiner als 1 oder grösser als die Anzahl der Statementnummern in der Statementnummernliste ist. In den genannten Fällen wird das imperative Statement nach dem Kennwort ELSE ausgeführt.[159b]
Angewandt auf das obige Beispiel entfallen zwei IF-Anweisungen, und es ergibt sich nachfolgender Programmabschnitt.

```
  1 REM N = KLEINSTER WERT VON I
  2 REM M = GROESSTER WERT VON I
  3 REM K = ANZAHL DER STATEMENTNUMMERN JE BEDINGTEN GOTO-STATEMENT
  6 LET I = I-N+1
  7 GOTO 10, 20, 30, . . . . . ON (I-1)/K+1 ELSE GOTO 999
 10 GOTO 100, 110, 120, . . . .  ON I
 20 GOTO 200, 210, 220, . . . .  ON I-K
 30 GOTO 300, 310, 320, . . . .  ON I-2*K
 40 GOTO 400, 410, 420, . . . .  ON I-3*K
   ....
   ....
999 PRINT "SPRUNGFEHLER"
   ....
   ....
```

Als imperatives Statement kann eine ausführbare BASIC-Elementaranweisung auftreten, die kein bedingtes GOTO-, IF- oder END-Statement

159a ANSI BASIC, p. 63.
159b ANSI BASIC, p. 64–65: "If the on-goto-statement contains an ELSE clause, and the value of the index in the on-goto-statement is less than one or greater than the number of line-numbers in the list, then the imperative-statement following the ELSE shall be executed."

sein darf; unzulässig ist auch der Beginn einer Programmschleife mit einem FOR- oder Start LOOP-Statement.
Das komplexe bedingte ON-GOTO-ELSE-Statement kann stets durch elementare BASIC-Anseisungen ersetzt werden, falls ein BASIC-Compiler diese Statementform nicht zulässt. So ist beispielsweise das Statement

```
10 ON I GOTO 10,20,30,40,50,60 ELSE GOTO 999
```

durch die Anweisungsfolge

```
10 IF I LE 0 GOTO 999
20 IF I GT 6 GOTO 999
ON I GOTO 10,20,30,40,50,60
   ....
   ....
```

ersetzbar.
Vom FOSBIC-Compiler werden bei Programmkompilierung erkannte Fehler wie folgt ausgewiesen:
Kennwortfehler bezüglich GOTO, ON oder OF:

```
          YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

Unzulässige Statementnummer:

```
          SORRY - I CAN ONLY GO TO LINES WITH NUMBERS BETWEEN 1 AND  9999
SEE BASIC TEXTBOOK  A 332.41
```

Während des Programmablaufs können zwei Fehlermeldungen auftreten.
Fehlende Statementnummer:

```
PROGRAM STOPPED -- I CANNOT GO TO   12 BECAUSE IT IS NOT THERE
                   ERROR CODE=  2
              ***** SEE BASIC TEXTBOOK =  A 332.41  *****
              ***** I WAS AT LINE NUMBER   10
```

Arithmetischer Ausdruck kleiner 1 oder grösser als die Anzahl der Statementnummer:

```
PROGRAM STOPPED --
THE VALUE OF THE EXPRESSION IN A COMPUTED GOTO OR GOSUB CANNOT BE
                USED TO SELECT A STATEMENT NUMBER - VALUE =          24
                ERROR CODE= 41
              ***** SEE BASIC TEXTBOOK =  A332.412  *****
              ***** I WAS AT LINE NUMBER    0
```

332.42 IF-Statements

Nach ANSI Minimal BASIC ist nur das einfache IF-Statement standardisiert[160], während nach ANSI BASIC zusätzliche Varianten des IF-Statements in Form des IF- und SELECT-Blocks zulässig sind.[160a]

160 ANSI Minimal BASIC, p. 16–17.
160a ANSI BASIC, p. 69: "An if-statement allows for the conditional execution

Da der SELECT-Block nur von wenigen BASIC-Compilern akzeptiert wird und immer durch eine Folge elementarer BASIC-Anweisungen ersetzt werden kann, wird hier auf seine Darstellung verzichtet. Für IF-Statements gilt demnach:

⟨IF-Statement⟩ : = ⟨Einfaches IF-Statement⟩ | ⟨IF-Block⟩
172 165 173 175a

332.421 Einfache IF-Statements

Jedes BASIC-konforme IF-Statement bewirkt, dass ein Programm mit dem unmittelbar folgenden Statement fortgesetzt wird, sofern der im IF-Statement spezifizierte Vergleich nicht zutrifft. Falls der Vergleich wahr ist, wird die im IF-Statement angegebene Anweisung voll ausgeführt. Es lassen sich folgende Formen des einfachen IF-Statements unterscheiden:

⟨Einfaches IF-Statement⟩
173 172

IF	⟨Vergleich⟩ 174	THEN	⟨Statementnummer⟩ 71 |		
IF	⟨Vergleich⟩ 174	GOTO	⟨Statementnummer⟩ 71 |		
IF	⟨Vergleich⟩ 174	THEN	⟨Statement-nummer⟩ 71	ELSE	⟨Statement-nummer⟩ 71 |
IF	⟨Vergleich⟩ 174	GOTO	⟨Statement-nummer⟩ 71	ELSE	⟨Statement-nummer⟩ 71 |
IF	⟨Vergleich⟩ 174	⟨Imperatives Statement⟩ 175 |			
IF	⟨Vergleich⟩ 174	THEN	⟨Imperatives Statement⟩ 175 |		
IF	⟨Vergleich⟩	THEN	⟨Imperatives Statement⟩ 175	ELSE	⟨Imperatives Statement⟩ 175

of a single imperative-statement or for the execution of one of two alternative imperative-statements. ... An if-block allows for the conditional execution of a sequence of lines or for the execution of one to two alternative sequences of lines. ... A select-block generalizes an if-block by allowing for the conditional execution of any one of a number of alternative sequences of lines."

Für den Vergleich gilt allgemein

$$\underset{174\ \ 173|175a}{\langle \text{Vergleich} \rangle} := \underset{54}{\left\langle \begin{matrix}\text{Komparativer}\\ \text{Ausdruck}\end{matrix} \right\rangle} \mid \underset{53}{\left\langle \begin{matrix}\text{Logischer}\\ \text{Ausdruck}\end{matrix} \right\rangle}$$

Der in IF-Statements integrierte Vergleich reduziert sich bei der Mehrzahl der BASIC-Compiler auf komparative Ausdrücke; vgl. Tabelle 15. Die in den Vergleich einzubeziehenden Ausdrücke können – unter Einhaltung der entsprechenden Bildungsregeln – beliebig strukturiert sein.

Die erste Form des einfachen IF-Statements in Gleichung 173 entspricht ANSI Minimal BASIC.[160b] Nach ANSI BASIC sind der erste, dritte, sechste und siebte Typ des IF-Statements zulässig.[160c] IF-Statements der zweiten und vierten Form entsprechen bis auf das Kennwort GOTO, das anstelle von THEN tritt, vollständig dem ersten bzw. dritten Typ.
Der fünfte Typ des einfachen IF-Statements entspricht in seiner logischen Struktur dem sechsten; es fehlt lediglich das Kennwort THEN.[160d] Sobald als imperatives Statement die Anweisung GOTO ⟨Statementnummer⟩ im Typ 5 verwendet wird, geht daraus die zweite Form der einfachen IF-Anweisung hervor. Nach Typ 6 ergibt sich die, im Sinne von ANSI BASIC ebenfalls zulässige, Statementform

IF ⟨Vergleich⟩ THEN GOTO ⟨Statementnummer⟩

Nach ANSI BASIC sind auch IF-Statements in der Form IF-THEN-ELSE zulässig. Bei ihnen wird, falls der Vergleich zutrifft, die Anweisung nach THEN ausgeführt oder, falls der Vergleich nicht zutrifft, das imperative Statement nach ELSE abgearbeitet.
Man bezeichnet das IF-THEN-ELSE-Statement auch als zweiseitig bedingtes IF-Statement. Es lässt sich grundsätzlich durch zwei einseitig bedingte einfache IF-Statements ersetzen. So entspricht beispielsweise die Anweisungsfolge

```
10 IF A = B THEN GOTO 200 ELSE LET N=0
20 GOTO 300
   ....
   ....
```

der Anweisungsfolge

160b ANSI Minimal BASIC, p. 16–17.
160c ANSI BASIC, p. 63–64, 69–70.
160d Zahlreiche Kleincomputersysteme lassen diese Form des einfachen IF-Statements zu; vgl. Nixdorf, System-Software Business-BASIC, S. 10.54–10.57.

```
10 IF A = B GOTO 200
15 LET N=0
20 GOTO 300
   ....
   ....
```

Allgemein kann das zweiseitige If-Statement immer durch die beiden einseitigen IF-Anweisungen

IF ⟨Vergleich⟩ THEN ⟨Imperatives Statement⟩

und

IF NOT ⟨Vergleich⟩ THEN ⟨Imperatives Statement⟩

substituiert werden.[160e]

Die imperativen Statements umfassen eine Menge elementarer BASIC-Anweisungen, die nach ANSI BASIC alle ausführbaren Anweisungen, mit Ausnahme der bedingten und der zu einer Programmschleife führenden Anweisungen enthält.[160f]

⟨Imperatives Statement⟩ : =
175 170a|173|194

⟨READ-Statement⟩ | ⟨INPUT-Statement⟩ | ⟨RESTORE-Statement⟩ |
115 117 118
⟨PRINT-Statement⟩ | ⟨PRINT USING-Statement⟩ | ⟨LET-Statement⟩ |
126 129 158
⟨Unbedingtes GOTO-Statement⟩ | ⟨STOP-Statement⟩ |
169 186
⟨Unbedingtes GOSUB-Statement⟩ | ⟨EXIT-Statement⟩
192 185

Bezüglich der zulässigen imperativen Statements bestehen zwischen den einzelnen BASIC-Compilersystemen erhebliche Unterschiede. Unzulässig sind jedenfalls, auch nach ANSI BASIC, bedingte GOTO- und GOSUB-Statements, alle Formen des IF-Statements, FOR- und Start LOOP-Statement, END-Statement.

160e Die Tendenz, mehrere elementare und einfache Anweisungen zu einer elementaren, aber komplexeren, Anweisung zusammenzufassen, lässt sich im Rahmen von ANSI BASIC auch bezüglich der ON GOTO- und ON GOSUB-Statements feststellen; vgl. Abs. 332.41 und 332.462.

160f ANSI BASIC, p. 12–13.

Unabhängig von den einem Vergleich zugrundeliegenden Ausdruckstypen lässt sich der für einseitig bedingte IF-Statements charakteristische Programmablauf in der aus Abbildung 27 ersichtlichen Weise darstellen.

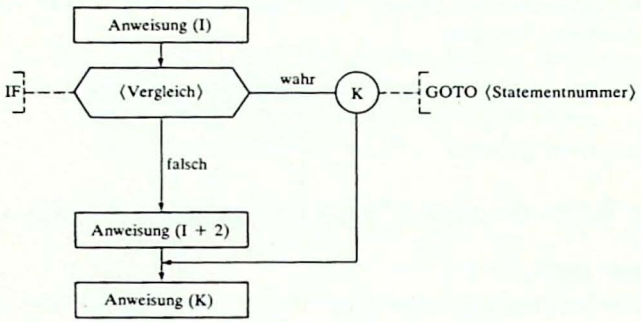


Abbildung 27. Flussdiagramm zum IF-Statement

Nach dieser Darstellung werden die Anweisungen (I) und (K) immer, die Anweisung (I + 2) nur bei nicht zutreffendem Vergleich, ausgeführt. Vgl. hierzu auch den folgenden Programmausschnitt, in Verbindung mit dem auf S. 221 angeführten Programm.

```
     ....
15   IF S GE 100 GOTO 30
20   GOTO 10
30   END
```

Die IF-Statements schaffen die Möglichkeit zur effizienten Codierung von Programmschleifen und vermitteln damit BASIC ein wesentliches Charakteristikum höherer Programmiersprachen.
Beispielsweise kann das folgende Flussdiagramm, was die Bildung der Summe aller ungeraden Zahlen von 1 bis 100 beinhaltet,

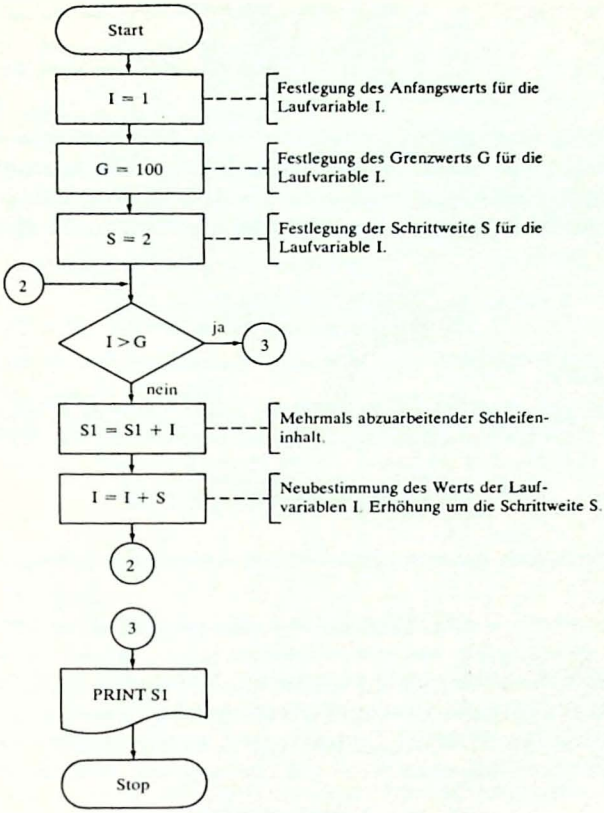


Abbildung 28. Flussdiagramm zur Bildung der Summe ungerader Zahlen

in das folgende, nur wenige Statements umfassende Programm,

```
   LET I = 1
   LET G = 100
   LET S = 2
2  IF I GT G GOTO 3
   LET S1 = S1 + I
   LET I = I + S
   GOTO 2
3  PRINT S1
   END
```

umgesetzt werden.[161]
Die Schleifenverarbeitung führt, wie der folgende Programmausschnitt zeigt,

161 Vgl. auch S. 226.

```
100  IF ABS(A) < 10 GOTO 900
110  IF ABS(A) = 10 GOTO 200
120  IF ABS(A) = 11 GOTO 300
130  IF ABS(A) = 12 GOTO 400
140  IF ABS(A) = 13 GOTO 500
150  IF ABS(A) > 13 GOTO 900
```

dann zu etwas uneleganten Strukturen, wenn IF-Statements in direkter Folge nacheinander auftreten. Dann ist nach Möglichkeit anstelle von IF- mit bedingten GOTO-Statements zu arbeiten.[162] Beispielsweise könnte der vorstehende Programmausschnitt äquivalent durch die beiden Statements

```
     IF ABS(A) < 10 GOTO 900
100  IF ABS(A) > 13 GOTO 900
110  GOTO 200,300,400,500, ON ABS(A)
```

ersetzt werden.
Die bisher verwendeten IF-Statements sind vollumfänglich FOSBIC-konform. Dies trifft auch für komplexere Statements der Form

```
100  IF A < B THEN 120
110  IF A**2 GT SIN(ABS(C)) GOTO 900
120  ...
```

zu. Dagegen würde das den vorstehenden Anweisungen äquivalente Statement

```
100  IF A≥ B AND A**2 ≤ SIN(ABS(C)) GOTO 900
```

vom FOSBIC-Compiler nicht verarbeitet. Entsprechendes gilt auch für IF-Statements mit logisch-komparativen Ausdrücken.[163]
Dagegen lässt der FOSBIC-Compiler den Vergleich alphanumerischer Ausdrücke via IF-Statements zu, wie das folgende Beispiel zeigt.

```
200  IF A$ = ≠ENDE≠ GOTO 999
```

Damit entspricht der FOSBIC-Compiler bezüglich der Verarbeitung von IF-Statements vollständig den Anforderungen von ANSI Minimal BASIC.[164]
Der FOSBIC-Compiler erfasst bei der Programmkompilierung des IF-Statements folgende Fehler:
Kenntwortfehler bezüglich IF:

```
          YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

162 Vgl. Abs. 332.412.
163 Vgl. Abs. 324.3.
164 ANSI Minimal BASIC, p. 16–17.

Kennwortfehler bezüglich THEN oder GOTO:

```
          I CANNOT FIND THEN OR GOTO
SEE BASIC TEXTBOOK  A 332.42
```

Fehlender oder falscher Vergleich:

```
          THE IF STATEMENT ABOVE IS HOPELESS
SEE BASIC TEXTBOOK  A 332.42

          I CANNOT ACCEPT AN IF STATEMENT WITHOUT A COMPARISON
SEE BASIC TEXTBOOK  A 332.42
```

Fehlende oder falsche Statementnummer:

```
          THEN OR GOTO MUST BE FOLLOWED BY A LINE NUMBER
SEE BASIC TEXTBOOK  A 332.42
```

Fehler in einem arithmetischen Ausdruck des komparativen Ausdrucks (Vergleich): Vgl. S. 100–101.
Während der Programmausführung kann die Fehlermeldung

```
PROGRAM STOPPED -- I CANNOT GO TO   12 BECAUSE IT IS NOT THERE
                   ERROR CODE=  2
              ***** SEE BASIC TEXTBOOK =  A 332.41  *****
              ***** I WAS AT LINE NUMBER    0
```

auftreten.

3.332.422 IF-Block

Der IF-Block unterliegt nur der ANSI BASIC-Norm.[164a] Im Gegensatz zum einfachen IF-Statement werden durch den Vergleich im IF-Block nicht nur zwei Statements in eine unterschiedliche Ablauffolge getrennt, sondern eine Anweisungsfolge in zwei Blöcke unterteilt, die in Abhängigkeit vom Vergleichsergebnis (wahr/falsch) getrennt durchlaufen werden.

Die Struktur eines IF-Blocks ist logisch identisch mit dem IF-THEN-ELSE-Statement. Nach ANSI BASIC gilt:

⟨IF-Block⟩ : =
175a 172

IF ⟨Vergleich⟩ THEN
174
 	
Folge beliebiger BASIC-Statements ohne END- und ohne data-Statements
 	
ELSE
 	
Folge beliebiger BASIC-Statements ohne END- und ohne data-Statements
 	
END IF

164a ANSI BASIC, p. 69–71.

Die Statementfolge zwischen den Kennworten IF und ELSE wird durchgeführt, falls der Vergleich zutrifft, andernfalls geht der Programmablauf direkt auf die Statementfolge zwischen den Kennworten ELSE und END IF über.[164b]
Das folgende Beispiel zeigt einen Programmausschnitt mit einem IF-Block.

```
 5  REM BEISPIEL ANSI BASIC P.70
10  IF X=INT(X) THEN
20    PRINT X;" IS AN INTEGER"
30  ELSE
40    PRINT X;" IS NOT AN INTEGER"
50  END IF
```

332.43 FOR/NEXT-Statements

Die mehrfache Abarbeitung von Programmabschnitten lässt sich durch FOR/NEXT-Statements besonders effizient regeln.[165]
FOR/NEXT-Statements treten paarweise auf und begrenzen einen spezifischen, zur mehrfachen Abarbeitung vorgesehenen Programmabschnitt. Dieser wird als Schleifeninhalt bezeichnet, wobei das FOR-Statement den Anfang und das NEXT-Statement das Ende der Programmschleife angeben.
FOR- und NEXT-Statements weisen generell die Form

⟨FOR-Statement⟩ : =
176 165

FOR ⟨Lauf-variable⟩ = ⟨Lauf-bereichs-anfang⟩ TO ⟨Lauf-bereichs-grenze⟩ STEP ⟨Schritt-weite⟩
178 179 180 181

164b ANSI BASIC, p. 71: "If the value of the relational-expression in an if-line is true, then the corresponding then-block shall be executed. If the value of the relational-expression is false, then the else-block ... shall be executed. After the appropriate then-block or else-block ... has been executed, execution shall continue at the line following the end-if-line."

165 ANSI Minimal BASIC, p. 17: "The for-statement and next-statement provide for the construction of loops."

⟨NEXT-Statement⟩ : =
177 165

NEXT ⟨Laufvariable⟩
178

auf, wobei zwischen dem FOR-Statement und dem zugehörigen NEXT-Statement eine beliebige Zahl von BASIC-Statements liegen kann. Zulässig sind alle Statementarten, ausser dem END-Statement.

⟨Laufvariable⟩ : = ⟨Einfache numerische Variable⟩
178 176|177 37|38
⟨Laufbereichsanfang⟩ : = ⟨Arithmetischer Ausdruck⟩
179 176 49
⟨Laufbereichsgrenze⟩ : = ⟨Arithmetischer Ausdruck⟩
180 176 49
⟨Schrittweite⟩ : = ⟨Arithmetischer Ausdruck⟩
181 176 49

Zusammengehörige FOR- und NEXT-Statements müssen die gleiche einfache numerische Variable als Laufvariable (control variable) aufweisen.[166] Sie nimmt zunächst den Laufbereichsanfangswert an und wird nach jeder vollständigen Abarbeitung des Schleifeninhalts jeweils um die sogenannte Schrittweite verändert. Sobald der Wert der Laufvariablen den Laufbereichsgrenzwert überschreitet, wird die Programmbearbeitung mit der unmittelbar auf das schleifenbegrenzende NEXT-Statement folgenden Anweisung fortgesetzt.

Dem FOR-Statement entspricht mithin die Anweisungsfolge

LET A = Laufbereichsanfang
LET G = Laufbereichsgrenze
LET S = Schrittweite
LET I = A

⟨Statement-nummer-i⟩ IF (I − G)*SGN(S) > 0 THEN ⟨Statement-nummer-j⟩

und dem NEXT-Statement die Anweisungsfolge

LET I = I + S
GOTO ⟨Statementnummer-i⟩

166 Vgl. Garland, Dartmouth BASIC, p. 37.

und

⟨Statement-nummer-j⟩ ⟨Nächstfolgendes BASIC-Statement ausserhalb der FOR/NEXT-Schleife⟩

Die folgende Abbildung 29 zeigt die Flowchartstruktur zur obigen Anweisungsfolge.

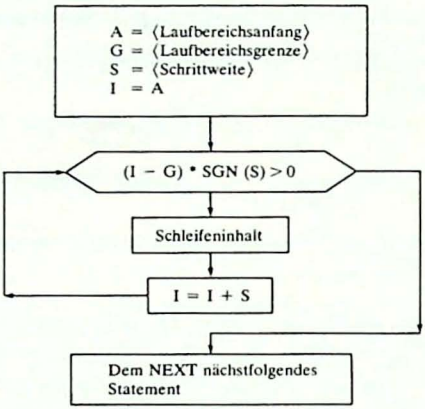


Abbildung 29. Flussdiagramm zum FOR/NEXT-Statement.

Das dem FOR/NEXT-Statement entsprechende vereinfachte Flowchartsymbol zeigt Abbildung 7 auf Seite 52.
Eine analoge Struktur weist auch das – bei Behandlung des IF-Statements angeführte – Programm zur Berechnung der Summe aller ungeraden Zahlen von 1 bis 100 auf.[167] Bei Verwendung von FOR/NEXT-Statements würde dieses Programm die Struktur

```
FOR I = 1 TO 100 STEP 2
LET S = S + I
NEXT I
PRINT S
END
```

erhalten.
Falls die Schrittweite +1 beträgt, braucht sie im FOR-Statement nicht explizit angeführt zu werden; das FOR-Statement lässt sich diesfalls etwas vereinfacht in der Form

167 Vgl. S. 221.

⟨FOR-Statement⟩ : =
182 165

FOR	⟨Laufvariable⟩ 178	=	⟨Laufbereichsanfang⟩ 179	TO	⟨Laufbereichsgrenze⟩ 180	

darstellen.
Vermerkt sei, dass alle im FOR-Statement erscheinenden arithmetischen Ausdrücke positive oder negative Werte annehmen und im I- oder F-Format definiert sein können. Mithin darf in BASIC auch mit negativen Schrittweiten gearbeitet werden.
Obwohl keine Ganzzahligkeit der Laufvariablen, der Laufbereichsgrenze und der Schrittweite vorgeschrieben wird, sollte der Programmierer dennoch auf Ganzzahligkeit – insbesondere bei der Schrittweite – achten. Vgl. hierzu das folgende Beispiel mit Schrittweite 0.1 und Laufbereich durch den Nullpunkt.

```
10 FOR I=-1 TO 1 STEP 0.1
20 PRINT I,
30 NEXT I
40 END

-1            -.9           -.8           -.7           -.6
 -.5          -.4           -.3           -.2           -.1
2.72848E-11   0.1           0.2           0.3           0.4
 0.5          0.6           0.7           0.8           0.9
```

Aus diesem Beispiel ist zu erkennen, dass die Laufbereichsgrenze von 1 nicht erreicht wird und die Schleife schon bei einem Laufvariablenwert von 0.9 abgebrochen wird. Ebenso trifft die Laufvariable nicht exakt den Wert Null, was durch die Art der computerinternen Datendarstellung bedingt ist.[168] Durch Einfügung eines LET-Statements innerhalb der Schleife und Verwendung einer ganzzahligen Schrittweite kann eine exakte Abarbeitung der Schleife und damit ein genaues Erreichen der gewünschten Zahlenwerte erzielt werden. Vgl. hierzu das folgende Programm.

```
10 FOR I=-10 TO 10
20 LET I1=I/10
30 PRINT I1,
40 NEXT I
50 END

-1            -.9           -.8           -.7           -.6
 -.5          -.4           -.3           -.2           -.1
 0.0          0.1           0.2           0.3           0.4
 0.5          0.6           0.7           0.8           0.9
 1
```

168 ANSI Minimal BASIC, p. 18: "Where arithmetic is approximate (as with decimal fractions in a binary machine), the loop wil be executed within the limits of machine arithmetic."

Die vorstehenden Beispiele zeigen auch, dass der temporäre Wert der Laufvariablen in schleifeninternen Statements beliebig verwendet – insbesondere auch ausgedruckt – werden kann.
Die mit der Verwendung von FOR/NEXT-Statements verbundenen programmiertechnischen Möglichkeiten werden weiterhin dadurch ausgeweitet, dass der temporäre Wert der Laufvariablen im Zuge der Schleifenabarbeitung modifizierbar ist. Ein Beispiel hierzu vermittelt das folgende Programm.

```
10 FOR I=1 TO 100 STEP 2
20 PRINT I,
30 LET I=I+10
40 NEXT I
50 END

 1          13          25          37          49
61          73          85          97
```

Programmiertechnisch wertvoll ist auch die Möglichkeit, der Laufvariablen die erforderlichen Start- und Grenzwerte sowie die Schrittweite schleifenextern – durch entsprechende READ- oder LET-Statements – zuzuweisen. Vgl. hierzu das folgende Beispiel.

```
10  LET I1 = -5
20  LET G = 10
30  FOR I = I1 TO G
40    PRINT I*3
50  NEXT I
60  END
```

Im Gegensatz zur FORTRAN und COBOL lässt der FOSBIC-Compiler auch Sprünge von ausserhalb in eine Schleife und von einer Schleife in eine andere Schleife zu, soweit beide Schleifen nicht verschachtelt sind. Vgl. hierzu das folgende Beispiel, bei dem der Schleifenanfang durch Überspringen des FOR-Statements auf −5 gesetzt und die Schleife entsprechend abgearbeitet wird.

```
10 LET I=-5
20 GOTO 40
30 FOR I=1 TO 10
40 LET X=I*3
50 PRINT I,X
60 NEXT I
70 END

-5          -15
-4          -12
-3           -9
-2           -6
-1           -3
 0.0          0.0
```

Einzelne BASIC-Compiler lassen auch Schrittweiten von Null zu;[169] vgl.

169 ANSI Minimal BASIC, p. 37: "There is to be no special error condition for an increment of zero since there is little reason to single out this particular infinite loop as an error (and not others) and since to do so would increase

Tabelle 15. Die Programmsteuerung muss diesfalls – zur Vermeidung einer unendlichen Schleife – durch entsprechend strukturierte schleifeninterne IF-Statements erfolgen, wie das folgende Beispiel zeigt.

```
10  READ S
20  DATA 1, 2, 0.0.
30  FOR I = 1 TO 10 STEP S
40    PRINT I
50    IF S EQ 0 GOTO 80
60  NEXT I
70  GOTO 10
80  PRINT #SCHRITTWEITE GLEICH NULL#,S
90  END
```

Vermerkt sei, daß die aus Abbildung 29 ersichtliche Art der Abarbeitung von FOR-Schleifen für die meisten BASIC-Compiler zutrifft. In wenigen Fällen wird zunächst der Schleifeninhalt (wenigstens einmal) abgearbeitet und erst nachher der Test auf Erreichung der Laufbereichsgrenze durchgeführt.[170] Diese Verarbeitungsweise findet sich u. a. bei den Computersystemen vom Typ GE 400 und DEC PDP-8.[171]
FOR/NEXT-Statements bieten auch die Möglichkeit zur Verarbeitung verschachtelter Schleifen (nested loops), wobei generell nur die beiden nachstehenden Schleifentypen zulässig sind.

```
FOR I = ...
  FOR J = ...
    FOR K = ...
    NEXT K
  NEXT J
NEXT I
```

```
FOR I = ...
  FOR J = ...
  NEXT J
  FOR K = ...
  NEXT K
Next I
```

executing time by requiring the increment value to be tested prior to executing a loop."

170 Terry M. Walker. Introduction to Computer Science: BASIC Language Programming. Boston 1973, p. 83, 216.

171 General Electric. GE-400 Series. BASIC Language for Time Sharing Programming System. Reference Manual Phoenix 1968, p. 25. Digital Equipment Corporation. Programming Languages. 2nd ed. Maynard 1972, p. 12.24: "If the value of the counter variable is originally set equal to the terminal value, the loop will execute once, regardless of the STEP value. If the starting value is beyond the terminal value, the loop will also execute only once."

Sich überlappende Schleifen der Art

```
┌FOR I = ...
│
│  ┌FOR J = ...
│  │
└NEXT I
   │
   └NEXT J
```

werden von den meisten BASIC-Compilern unter Warnung des Programmierers als

```
┌FOR I = ...
│
│  ┌FOR J = ...
│  │
│  └NEXT J
│
└NEXT I
```

interpretiert. Dies trifft auch für den FOSBIC-Compiler zu, wobei für das vorstehende Beispiel zwei Warnungen ausgegeben würden,
nach NEXT I:

```
        I EXPECTED YOU TO MENTION VARIABLE J  -- I WILL ASSUME THAT YOU DID
SEE BASIC TEXTBOOK  A 332.43
```

nach NEXT J:

```
        I EXPECTED YOU TO MENTION VARIABLE I  -- I WILL ASSUME THAT YOU DID
SEE BASIC TEXTBOOK  A 332.43
```

Verschachtelte Schleifen mit gleichen Laufvariablen sind nicht zulässig; sie werden vom FOSBIC-Compiler bei der Programmkompilierung nicht erkannt, führen aber bei der Programmabarbeitung generell zu sinnlosen Ergebnissen.[172]
Die höchstzulässige Zahl verschachtelter Schleifen ist compilerabhängig beschränkt. Vgl. Tabelle 15.
Verschachtelte Schleifen sind insbesondere zur Matrizenverarbeitung erforderlich. Vgl. hierzu das folgende Multiplikationsprogramm, mit

172 Garland, Dartmouth BASIC, p. 38: "Nested for loops with the same index variable will not work."

$$c_{ik} = \sum_{r=1}^{n} a_{ir} b_{rk}$$

```
10   REM MATRIZENMULTIPLIKATION
20   DIM A(3,4),B(4,3),C(3,3)
30   READ N1,M1,N2,M2,N3,M3
40   DATA 3,4,4,3,3,3
50   REM PRUEFEN DER INDEXBEREICHE
60   IF N1 <> M2 GOTO 360
70   IF N1 > N3 GOTO 360
80   IF M2 > M3 GOTO 360
90   REM EINLESEN DER MATRIZENELEMENTE
100  FOR I=1 TO N1
110    FOR K=1 TO M1
120      READ A(I,K)
130    NEXT K
140  NEXT I
150  FOR I=1 TO N2
160    FOR K=1 TO M2
170      READ B(I,K)
180    NEXT K
190  NEXT I
200  REM AUSFUEHRUNG DER MATRIZENMULTIPLIKATION
210  FOR I=1 TO N3
220    FOR K=1 TO M3
230      FOR R=1 TO M1
240        LET C(I,K)=C(I,K)+A(I,R)*B(R,K)
250      NEXT R
260    NEXT K
270  NEXT I
280  REM AUSGABE DER ERGEBNISSE
290  FOR I=1 TO N3
300    FOR K=1 TO M3
310      PRINT C(I,K),
320    NEXT K
330    PRINT
340  NEXT I
350  GOTO 420
360  PRINT #FEHLER IM INDEXBEREICH#
370  PRINT N1,M1
380  PRINT N2,M2
390  PRINT N3,M3
400  DATA 1,2,3,4,5,6,7,8,9,10,11,12
410  DATA 1,2,3,4,5,6,7,8,9,10,11,12
420  END

*** EVERYTHING SEEMS OK -- LET#S GO AHEAD

     PERCENT OF AVAILABLE STORAGE USED                23.278
     PERCENT OF AVAILABLE DATA STORAGE USED            9.091
     PERCENT OF AVAILABLE NUMBERED STATEMENTS USED    12.353

      70              80              90
     158             184             210
     246             288             330
```

Da den Laufvariablen bei verschachtelten Schleifen über externe READ- oder LET-Statements beliebige Start- und Grenzwerte sowie Schrittweiten zugewiesen werden können, besteht – wie der folgende Programmausschnitt zeigt – auch die Möglichkeit zur Ausgabe von Teilbereichen einer Matrize.

```
280  REM TEILAUSGABE DER ERGEBNISSE DER MATRIZENMULTIPLIKATION
281  REM GEMAESS VORIGEM BEISPIEL
282  READ Z1, S1, Z2, S2, Z3, S3
283  REM Z1 = ANFANGSZEILE DER AUSGABE
284  REM Z2 = ENDZEILE DER AUSGABE
285  REM Z3 = ZEILENSCHRITTWEITE
286  REM S1 = ANFANGSSPALTE DER AUSGABE
287  REM S2 = ENDSPALTE DER AUSGABE
288  REM S3 = SPALTENSCHRITTWEITE
```

```
289  DATA 2,2,3,3,1,1
290  FOR I = Z1 TO Z2 STEP Z3
300      FOR K = S1 TO S2 STEP S3
310          PRINT C(I,K);
320      NEXT K
330      PRINT
340  NEXT I
350  GOTO 420

 184 210
 288 330
```

Der FOSBIC-Compiler moniert bei der Programmkompilierung feststellbare Fehler wie folgt:
Kennwortfehler bei FOR oder NEXT:

```
        YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

FOR-Statement unvollständig oder mit groben Fehlern in einzelnen Bereichen:

```
        THE FOR STATEMENT ABOVE IS HOPELESS
SEE BASIC TEXTBOOK  A 332.43
```

Indizierte Laufvariable:

```
        THE VARIABLE IN A FOR STATEMENT MUST BE UNSUBSCRIPTED
SEE BASIC TEXTBOOK  A 332.43
```

Fehlendes Zuordnungszeichen:

```
        I CANNOT FIND AN EQUAL SIGN IN THE FOR STATEMENT ABOVE
SEE BASIC TEXTBOOK  A 332.43
```

Fehlendes oder falsches TO:

```
        I CANNOT FIND A TO
SEE BASIC TEXTBOOK  A 332.43
```

Fehler im arithmetischen Ausdruck zur Bestimmung von Laufbereichsanfang, Laufbereichsende oder Schrittweite: vgl. S. 87–88.
Mehr FOR-Statements als NEXT-Statements:

```
        THERE ARE     0 FOR STATEMENTS WITHOUT NEXT STATEMENTS
SEE BASIC TEXTBOOK  A 332.43
```

Mehr NEXT-Statements als FOR-Statements:

```
        EITHER YOU FORGOT A FOR STATEMENT OR I REJECTED IT
SEE BASIC TEXTBOOK  A 332.43
```

Schleifenüberlappung, mit automatischer Umstellung der Laufvariablen:

```
      I EXPECTED YOU TO MENTION VARIABLE 00 -- I WILL ASSUME THAT YOU DID
SEE BASIC TEXTBOOK  A 332.43
```

Unzulässige Verschachtelungstiefe:

```
          I CAN ONLY HANDLE  20 FOR STATEMENTS WITHOUT NEXT STATEMENTS
SEE BASIC TEXTBOOK  A 332.43
```

Während des Programmablaufs wird moniert:
Schrittweite gleich Null:

```
PROGRAM STOPPED -- ZERO STEP SIZE IN A FOR STATEMENT
                   ERROR CODE= 18
              ***** SEE BASIC TEXTBOOK =  A 332.43  *****
              ***** I WAS AT LINE NUMBER  18
```

332.44 LOOP/EXIT-Statements

Neben den vorerwähnten FOR/NEXT-Statements werden nach ANSI BASIC die LOOP/EXIT-Statements zur Schleifenprogrammierung vorgesehen.[172a]
Der Anfang einer Programmschleife wird durch das Start LOOP-Statement, das Ende der Programmschleife durch das END LOOP-Statement angezeigt. Start LOOP- und END LOOP-Statements treten immer paarweise auf; für sie gilt:

⟨Start LOOP-Statement⟩ : =
183 165

LOOP

⟨END LOOP-Statement⟩ : =
184 165

END LOOP

Alle zwischen den genannten LOOP-Statements liegenden Anweisungen werden so lange durchlaufen, bis die Schleife über ein bedingtes oder unbedingtes GOTO-Statement oder über ein EXIT-Statement verlassen werden kann.
Das EXIT-Statement kann nur innerhalb einer LOOP-Schleife auftreten; es ist wie folgt strukturiert:

172a ANSI BASIC, p. 66: "Loops provide for the repeated execution of a sequence of statements. The textual beginning of a loop is indicated by the loop-statement LOOP, and the textual end by the end-loop-statement END LOOP. Execution of a loop terminates upon execution of an exit-statement."

⟨EXIT-Statement⟩
185 165

EXIT

Dem EXIT-Statement kommt der Charakter eines speziellen unbedingten GOTO-Statements zu, das als imperatives Statement vor allem in Verbindung mit IF-Statements Verwendung findet und bewirkt, dass das Programm mit dem nächsten Statement fortgesetzt wird, das dem zur LOOP-Schleife gehörenden END LOOP-Statement folgt.
LOOP-Schleifen können auch verschachtelt werden.
Die Anwendung einer LOOP-Schleife zeigt der folgende Programmabschnitt.

```
100  LOOP
110    PRINT "EINGABE EINER GANZZAHL ZWISCHEN 1 UND 10"
120    INPUT N
130    IF N>0 AND N<=10 AND N=INT(N) THEN EXIT
140  END LOOP
```

Das Beispiel verdeutlicht, dass eine LOOP-Schleife durch ihre Struktur zunächst endlos ist und nur über ein in der Programmschleife enthaltenes EXIT- (oder GOTO-)Statement verlassen werden kann.[172b]

332.45 STOP-Statements

Das STOP-Statement führt zu einem Abbruch des Programmablaufs[173]. Es stellt das logische Ende eines Programms dar und kann daher in einem Programm an beliebiger Stelle – auch mehrmals – erscheinen, ohne sich auf die Programmkompilation auszuwirken.
Das STOP-Statement weist nach ANSI Minimal BASIC die Form

⟨STOP-Statement⟩ : =
186 166

STOP

172b ANSI BASIC, p. 67: "If execution of a program reaches a [start] loop-line, then it shall proceed to the next line. If execution reaches an end-loop-line, the execution shall be continued at the associated [start] loop-line. An exit-statement, when executed, causes execution to be continued at the line following the end-loop-line of the smallest loop in which that exit-statement occurs."

173 ANSI Minimal BASIC, p. 17: "The stop-statement causes termination of the program."

auf; einzelne BASIC-Compiler lassen auch die Version

⟨STOP-Statement⟩ : =
187 166

STOP ⟨Stringinhalt⟩
34

zu, wobei diese Form insbesondere zur Kennzeichnung des benutzten Programmausgangs dient. Vgl. hierzu die folgende Darstellung.

```
10  REM PROGRAMMANFANG
20
30
    ...
90
100 IF A > 100 GOTO 200
110
120
    ...
180
190 GOTO 220
200 PRINT #FEHLER BEI A>100#,A
210 STOP
220 REM PROGRAMMFORTSETZUNG
230
240
    ...
290
300 IF B = 0 GOTO 400
310
320
    ...
380
390 GOTO 410
400 STOP
410 REM PROGRAMMFORTSETZUNG
420
430
    ...
590
600 STOP  PROGRAMMENDE
610 END
```

332.46 END-Statements

Das END-Statement bezeichnet das physikalische Ende einer als Einheit zu verarbeitenden Folge von BASIC-Statements und bewirkt den Abschluss der Programmkompilation ohne Berücksichtigung eventuell nachfolgender Statements.[174]
Neben der einfachen Form

⟨END-Statement⟩ : =
188 166

END

174 ANSI Minimal BASIC, p. 11: "The end-statement serves both to mark the physical end of the main body of a program and to terminate the execution of the program when encountered."

lassen einzelne Compiler auch die Variante

⟨END-Statement⟩ : =
189 166

END ⟨Stringinhalt⟩
34

zu.
Nach ANSI Minimal BASIC ist das END-Statement zur Kennzeichnung des Programmendes unbedingt erforderlich;[175] dies trifft auch für den FOSBIC-Compiler zu, ist aber nicht bei allen BASIC-Compilern notwendig.
Soweit von einem BASIC-Compiler auch data-Statements[176] verarbeitet werden sollen, müssen sie dem END-Statement folgen. Dabei bietet der FOSBIC-Compiler mit dem

⟨END-Statement⟩ : =
190 166

END LIST DATA

noch die Möglichkeit zur Auflistung dieser data-Karten. Vgl. hierzu das folgende Kurzprogramm.

```
10 PRINT ≠END-TEST≠
20 READ A, B, C, D$, E$, F$
30 PRINT A, B, C, D$, E$, F$
40 END LIST DATA

LIST OF DATA AFTER END STATEMENT

1,2,3,
≠DATA≠, ≠ENDE≠, ≠BASIC≠
*

*** EVERYTHING SEEMS OK -- LET≠S GO AHEAD

PERCENT OF AVAILABLE STORAGE USED                11.676
PERCENT OF AVAILABLE DATA STORAGE USED            0.000
PERCENT OF AVAILABLE NUMBERED STATEMENTS USED     1.176

END-TEST
        1              2              3      DATA          ENDE
BASIC
```

175 ANSI Minimal BASIC, p. 34: "The end-statement is required to be the physically last statement of a program in order to safeguard the inexperienced user from submitting an incomplete program for execution."
176 Vgl. Abs. 332.12.

Tabelle 16. Verfügbarkeit und Art von STOP-, END-, GOSUB- und RETURN-Statement

Prozessor	STOP-Statement	END-Statement	Unbedingtes GOSUB-Statement	Bedingtes GOSUB-Statement Verfügbarkeit[1]	ganzzahliger Teil des arithmet. Ausdrucks	Maximale Verschachtelungstiefe	END-Statement notwendig
ANSI Minimal BASIC	186	188	v	n		(10)	ja
Dartmouth BASIC-6	186	188	v	v	INT ⟨arithm. Ausdruck⟩	255	ja
UOM BASIC/ UBASIC	186	188	v	n		50	ja
IBM CALL BASIC	186/187	188/189	v	v	Abschneiden	47	ja
UWBIC	186	188	v	n		20	ja
FOSBIC	186/187	188/190/ 189	v	v	INT ⟨arithm. Ausdruck⟩	einstellbar (CD 3300 = 20)	ja

v = verfügbar
n = nicht verfügbar
Die Zahlenangaben beziehen sich auf die Nummern von Definitionsgleichungen.
1 Zur Verfügbarkeit einzelner Statementformen vgl. Weber und Türschmann, BASIC 1, 1. Aufl., S. 192; Weber, BASIC Programmier-Richtlinien für Austauschprogramme, S. 78–81.

Zur dritten Klasse der Programmsteuerungsanweisungen gehören die Statements GOSUB und RETURN. Sie dienen zum Aufruf von Unterprogrammen mit automatischem Rücksprung[177] und werden deshalb Programmkoppelungsanweisungen genannt.

332.47 GOSUB-Statements

GOSUB ist eine Sprunganweisung, die den Übergang zur ersten – durch die Statementnummer identifizierten – Anweisung eines Unterprogramms bewirkt. Der Beginn dieser Subroutine ist generell nur aus der im GOSUB-Statement als Sprungziel angegebenen Statementnummer erkennbar. Zweckmässigerweise fällt das Sprungziel allerdings auf ein REM-Statement, das eine kurze Umschreibung der Subroutine beinhaltet.
Analog zum GOTO-Statement unterscheidet man auch bei der GOSUB-Anweisung zwischen zwei verschiedenen Statementtypen.

⟨GOSUB-Statement⟩ : = ⟨Unbedingtes GOSUB-Statement⟩ | ⟨Bedingtes GOSUB-Statement⟩
191 167 192 193|194

332.471 Unbedingtes GOSUB-Statement
Mit dem unbedingten GOSUB-Statement wird – in Analogie zum unbedingten GOTO-Statement – jedenfalls auf das angegebene Sprungziel verzweigt.
Das unbedingte GOSUB-Statement weist die Formen

⟨Unbedingtes GOSUB-Statement⟩ : =
192 191

GOSUB ⟨Statementnummer⟩ | 71
GO SUB ⟨Statementnummer⟩ 71

auf. Beide Formen unterscheiden sich nur in der Schreibweise des Kennwortes GOSUB; beide Formen sind nach ANSI Minimal BASIC und ANSI BASIC zugelassen.[177a]
Eine Anwendung vermittelt der folgende Programmausschnitt.

177 ANSI Minimal BASIC, p. 16–17.
177a ANSI Minimal BASIC, p. 17; ANSI BASIC, p. 63–64.

```
10   REM PROGRAMMANFANG
20
30
   ...
80   IF M < 1 GOTO 300
90   IF M > 4 GOTO 300
100  IF M = 1 GOTO 200
110  IF M = 2 GOTO 220
120  IF M = 3 GOTO 240
130  IF M = 4 GOTO 260
200  GOSUB 1000
210  GOTO 300
220  GOSUB 2000
230  GOTO 300
240  GOSUB 3000
250  GOTO 300
260  GOSUB 4000
270  GOTO 300
280
290
300  REM HAUPTPROGRAMMFORTSETZUNG
310
   ...
990  GOTO 5000  ← ENDE DES HAUPTPROGRAMMS
1000 REM BEGINN DES ERSTEN UNTERPROGRAMMS
1010
   ...
1100 RETURN  AUS ERSTEM UNTERPROGRAMM
2000 REM BEGINN DES ZWEITEN UNTERPROGRAMMS
2010
   ...
2100 RETURN AUS ZWEITEM UNTERPROGRAMM
   ...
   ...
   ...
5000 END  ENDE DES HAUPTPROGRAMMS UND ALLER UNTERPROGRAMME
```

332.472 Bedingte GOSUB-Statements

Wie aus Tabelle 16 ersichtlich ist, lassen einzelne BASIC-Compiler auch bedingte (verzweigte) GOSUB-Statements zu. Sie entsprechen in Funktion und Struktur den bedingten GOTO-Statements.

⟨Bedingtes GOSUB-Statement⟩ : =
193 191

ON	⟨Arithmetischer Ausdruck⟩ 49	GOSUB	⟨Statement nummernliste⟩ 171	|
OF	⟨Arithmetischer Ausdruck⟩ 49	GOSUB	⟨Statement-nummernliste⟩ 171	|
GOSUB	⟨Statement-nummernliste⟩ 171	ON	⟨Arithmetischer Ausdruck⟩ 49	|
GOSUB	⟨Statement-nummernliste⟩ 171	OF	⟨Arithmetischer Ausdruck⟩ 49	

Bedingte GOSUB-Statements sind nach ANSI Minimal BASIC nicht vorgesehen; ANSI BASIC entspricht nur die ersterwähnte Form, wobei – analog zum unbedingten GOSUB-Statement – auch die Schreibweise GO SUB zugelassen wird.[177b]
Vom FOSBIC-Compiler werden alle Varianten verarbeitet.
Der Wert des in einem GOSUB-Statement erscheinenden arithmetischen Ausdrucks wird nötigenfalls durch Rundung oder Abschneiden des Dezimalteils ganzzahlig gemacht. Ist dieser Wert kleiner als 1 oder grösser als die Zahl der in der Statementnummernliste angegebenen Statementnummern, so wird – ohne Sprung in eines der angegebenen Unterprogramme – unmittelbar mit der auf das bedingte GOSUB-Statement folgenden Anweisung fortgefahren. Vgl. hierzu den folgenden Programmausschnitt, der inhaltlich mit dem für das unbedingte GOSUB-Statement angegebenen Beispiel übereinstimmt.

```
10   REM PROGRAMMANFANG
20
30
  ...
80   IF M < 1 GOTO 300
90   IF M > 4 GOTO 300
100  GOSUB 1000, 2000, 3000, 4000 ON M
300  REM PROGRAMM FORTSETZUNG WIE IM ORIGEM BEISPIEL
```

Analog zum bedingten GOTO-Statement normiert ANSI BASIC auch das bedingte GOSUB-Statement mit der ELSE-Erweiterung.

⟨Bedingtes GOSUB-Statement⟩ : =
194 191

ON	⟨Arithmetischer Ausdruck⟩	GOSUB	⟨Statement-nummernliste⟩	ELSE	⟨Imperatives Statement⟩	
	49		171		175	

Mit dieser Statementform wird die Fehlermöglichkeit, dass der ganzzahlige Wert des arithmetischen Ausdrucks kleiner als 1 oder grösser als die Anzahl der Statementnummern in der Statementnummernliste ist, aufgefangen und das imperative Statement ausgeführt.
Angewendet auf die beiden letzten Programmbeispiele ergibt sich eine Vereinfachung durch Einsparung der beiden IF-Statements.

```
 10 REM PROGRAMMANFANG (WIE OBIGE BEISPIEL ZUM ON GOSUB-STATEMENT)
 20
 30
  .....
 80 REM DIESES IF-STATEMENT ENTFAELLT NUN
 90 REM DIESES IF-STATEMENT ENTFAELLT EBENFALLS
100 ON M GOSUB 1000, 2000, 3000, 4000 ELSE GOTO 300
300 REM PROGRAMMFORTSETZUNG WIE IN DEN OBIGEN BEISPIELEN
```

177b ANSI BASIC, p. 63–65.

332.48 RETURN-Statement

Das RETURN-Statement stellt die letzte logische Anweisung einer Subroutine dar und bewirkt den Rücksprung in das Ausgangsprogramm. Dieses wird mit der dem GOSUB-Statement unmittelbar folgenden (ausführbaren) Anweisung fortgesetzt.
Neben dem einfachen, nach ANSI Minimal BASIC und ANSI BASIC vorgesehenen[177c],

⟨RETURN-Statement⟩ : =
197 167

RETURN

lassen einzelne Compiler auch die Alternativform

⟨RETURN-Statement⟩ : =
198 167

RETURN ⟨Stringinhalt⟩
34

zu; damit ergibt sich die Möglichkeit, das RETURN-Statement mit einem Kommentar – bestehend aus einem oder mehreren Grundsymbolen – zu versehen.
Auf Kleincomputeranlagen ist oft auch folgende Variante des RETURN-Statements verfügbar:

⟨RETURN-Statement⟩ : =
198a 167

RETURN ⟨Arithmetischer Ausdruck⟩
49

Die Struktur dieses RETURN-Statements unterscheidet sich von den früher erwähnten dadurch, dass dem Kennwort RETURN ein arithmetischer Ausdruck folgt. Der positive oder negative Ganzzahlwert dieses Ausdrucks gibt an, um wie viele Statements die Programmfortführung im aufrufenden Programm vor- oder rückwärts versetzt werden soll. Als Ausgangsposition dient die dem GOSUB-Statement folgende Anweisung.
Im speziellen wird durch die Anweisung

```
2010 RETURN -1
```

177c ANSI Minimal BASIC, p. 16–17; ANSI BASIC, p. 63–64.

bewirkt, dass der Rücksprung aus dem Unterprogramm erneut auf die das Unterprogramm aufrufende GOSUB-Anweisung führt.
Die letzterwähnte Variante des RETURN-Statements erlaubt eine besonders effiziente Anwendung der Unterprogrammtechnik, da im Unterprogramm selbst festgelegt werden kann, wo der Programmablauf im übergeordneten Programm fortgesetzt wird.

332.49 Unterprogrammtechnik mit GOSUB und RETURN

Die sich über entsprechende GOSUB- und RETURN-Statements ergebenden Verbindungen zwischen Basis- und Unterprogramm sind in Abbildung 30 für nicht verschachtelte und in Abbildung 31 für verschachtelte Subroutinen dargestellt. Jedes Basisprogramm kann ein oder mehrere GOSUB-Statements aufweisen, wobei das zu jedem GOSUB gehörende Unterprogramm den zwischen dem GOSUB-Sprungziel und dem entsprechenden RETURN-Statement liegenden Programmteil umfaßt.[178]
Aus den Abbildungen 30 und 31 wird deutlich, dass das aufgerufene Unterprogramm niemals unmittelbar nach dem aufrufenden GOSUB-Statement beginnen darf, da ansonsten mittels der RETURN-Anweisung erneut in das gerade durchlaufene Unterprogramm gesprungen wird und so eine endlose Schleife entsteht.

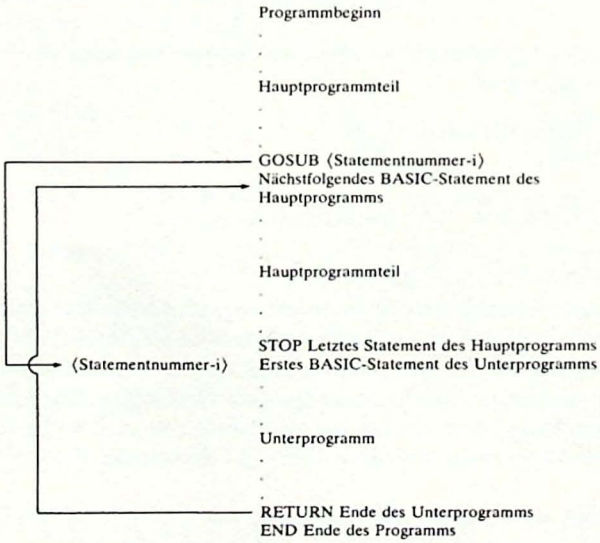


Abbildung 30. Unterprogrammtechnik mit GOSUB und RETURN

178 Vgl. auch ANSI Minimal BASIC, p. 16–17.

Die höchstzulässige Verschachtelungstiefe ist compilerabhängig. Nach ANSI Minimal BASIC sollten sich ursprünglich bis zu 10 GOSUB-Statements ohne dazwischenliegende RETURN-Statements folgen können; diese Regelung wurde aber nachträglich aufgehoben.[179] Der FOSBIC-Compiler lässt Verschachtelungen bis zu einer von der Implementierung abhängigen Tiefe zu. Vgl. Tabelle 16.

Die Anordnungsfolge der Subroutinen untereinander ist beliebig, da jedes Unterprogramm eine in sich abgeschlossene Befehlsfolge darstellt. Die logische Aufeinanderfolge der Subroutinen wird programmspezifisch durch die entsprechenden GOSUB-Statements erzeugt und eingehalten. Vermerkt sei, dass die Subroutinen auch STOP-Statements mitumfassen und damit unter bestimmten Bedingungen auch den Programmabschluss – ohne vorherigen Rücksprung in das Hauptprogramm – bewirken können.[180] Weiterhin kann jedes Unterprogramm bei Ersetzung des RETURN-Statements durch ein END-Statement verselbständigt werden, wodurch die Durchführung subroutinebezogener Programmtests erleichtert wird.

Vom FOSBIC-Compiler werden zu den GOSUB- und RETURN-Statements folgende Fehler ausgewiesen:

Kennwortfehler bei GOSUB oder RETURN:

```
        YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

Unzulässige Statementnummer:

```
        SORRY - I CAN ONLY GO TO LINES WITH NUMBERS BETWEEN 1 AND  9999
SEE BASIC TEXTBOOK  A 332.41
```

Fehler im arithmetischen Ausdruck eines bedingten GOSUB:
Vgl. Abs. 324.1.

RETURN-Statement ohne zugehöriges GOSUB-Statement:

```
PROGRAM STOPPED -- I CANNOT RETURN SINCE I DID NOT COME FROM A GOSUB
                   ERROR CODE=  3
              ***** SEE BASIC TEXTBOOK =  A 332.46  *****
              ***** I WAS AT LINE NUMBER    0
```

Unzulässige Verschachtelungstiefe:

```
PROGRAM STOPPED -- I CAN ONLY HANDLE  20 GOSUBS WITHOUT RETURNS
                   ERROR CODE=  1
              ***** SEE BASIC TEXTBOOK =  A 332.46  *****
              ***** I WAS AT LINE NUMBER    0
```

Die beiden letzterwähnten Fehler werden während der Programmausführung moniert.

179 Vgl. Weber und Türschmann, BASIC 1, 1. Aufl., S. 197.

180 ANSI Minimal BASIC, p. 17: "It is not necessary that equal numbers of gosub-statements and return-statements be executed before termination of the program."

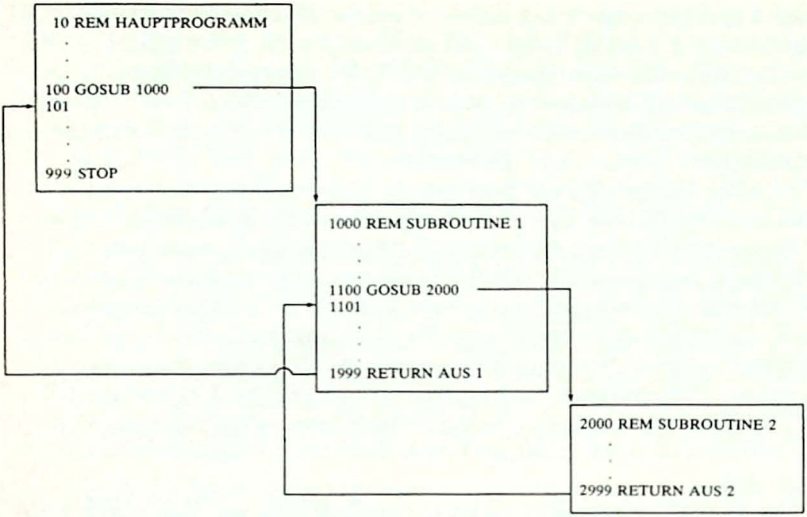


Abbildung 31. Verschachtelte Subroutinen

34 Programmaufbau

Die bisher besprochenen Elementaranweisungen bilden die Grundlage zur Erarbeitung jeglicher Art von BASIC-Programmen.
Mit Ausnahme des END-Statements kann jedes BASIC-Statement in einem Programm mehrmals erscheinen. Pro Statement ist eine (Programm-)Zeile erforderlich.[181] Eine Ausnahmeregelung besteht nur für das null-Statement, das unmittelbar an ein anderes (Elementar-)Statement angefügt werden kann.[182]
Bezüglich der BASIC-Programmstruktur gilt allgemein:

$$\underset{199}{\langle \text{BASIC-Programm} \rangle} := \underset{201}{\left\langle \begin{matrix}\text{Programm-}\\ \text{endteil}\end{matrix} \right\rangle} \mid \underset{200}{\left\langle \begin{matrix}\text{Programm-}\\ \text{hauptteil}\end{matrix} \right\rangle} \underset{201}{\left\langle \begin{matrix}\text{Programm-}\\ \text{endteil}\end{matrix} \right\rangle}$$

181 BASIC wird deshalb oft als zeilenorientierte Programmiersprache charakterisiert. Vgl. Abs. 14 und 326 und NNI Elementair BASIC, p. 25: "BASIC is een regel-georienteerde programmeertaal."
182 Vgl. Abs. 331.12.

$$\underset{200}{\langle \text{Programm-hauptteil} \rangle} \underset{199}{::=} \underset{203}{\langle \text{Programm-hauptteil-statement} \rangle} \mid \underset{200}{\langle \text{Programm-hauptteil} \rangle} \underset{203}{\langle \text{Programm-hauptteil-statement} \rangle}$$

$$\underset{201}{\langle \text{Programm-endteil} \rangle} \underset{199}{::=} \underset{188|189|190}{\langle \text{END-Statement} \rangle} \mid \underset{188|189|190}{\langle \text{END-Statement} \rangle} \underset{202}{\langle \text{data-Teil} \rangle}$$

$$\underset{202}{\langle \text{data-teil} \rangle} \underset{201}{::=} \underset{114}{\langle \text{data-Statement} \rangle} \mid \underset{202}{\langle \text{data-Teil} \rangle} \underset{114}{\langle \text{data-Statement} \rangle}$$

$$\underset{203}{\langle \text{Programm-hauptteil-statement} \rangle} \underset{200}{::=} \underset{76|205}{\langle \text{BASIC-Anweisung, nicht END-Statement, nicht data-Statement} \rangle}$$

Jedes Programm muss mindestens ein END-Statement enthalten. Das denkbar kürzeste BASIC-konforme – aber nicht zur Ausführung gelangende – Programm besteht nur aus einem END-Statement.[183] So wird beim FOSBIC-Compiler auch

```
.
END
.
```

als Programm akzeptiert und bei der Kompilierung mit folgenden Hinweis als fehlerfrei anerkannt:

```
*** EVERYTHING SEEMS OK — LET'S GO AHEAD

    PERCENT OF AVAILABLE STORAGE USED                10.162
    PERCENT OF AVAILABLE DATA STORAGE USED            0.000
    PERCENT OF AVAILABLE NUMBERED STATEMENTS USED     0.000
```

183 NNI Elementair BASIC, p. 25: "Een programma bestaat uit een of meer opdrachtregels. ... De laatste opdrachtregel bevat en END-opdracht."

4 BASIC-Anwendungsbeispiele

In den folgenden Abschnitten werden ausgewählte Probleme aus den Gebieten der Betriebswirtschaftslehre, der Volkswirtschaftslehre und der Statistik mit BASIC-Programmen gelöst, wobei nur die in diesem Band besprochenen Elementaranweisungen zum Einsatz gelangen[1].
Die zur Problembearbeitung erforderlichen theoretischen Grundkenntnisse werden beim Leser vorausgesetzt.
Die Darstellung der Problemstruktur und des Lösungswegs beschränkt sich auf ein Minimum und ist auf den Aufbau des jeweiligen BASIC-Programms abgestimmt.

41 Betriebswirtschaftslehre

411 Exponentielle Glättung

Die exponentielle Glättung[2] stellt ein einfaches Verfahren der Prognoserechnung dar, wobei aus Zeitreihenwerten auf Grund der Beziehung

$$y_t^{(1)} = \alpha \sum_{i=0}^{n} (1-\alpha)^i y_{t-i}$$

mit t = Periode und α = Glättungsfaktor gewichtet arithmetische Mittelwerte $y_t^{(1)}$ gebildet werden. Mittelwerte k-ter Ordnung errechnen sich nach der Gleichung

$$y_t^{(k)} = \alpha \sum_{i=0}^{n} (1-\alpha)^i y_{t-i}^{(k-i)}$$

Unter Verwendung der Daten von zwei aufeinanderfolgender Perioden und zweier Mittelwerte k und k-1 lässt sich die letzte Gleichung vereinfachen zu

$$y_t^{(k)} = \alpha y_t^{(k-1)} + (1-\alpha) y_{t-1}^{(k)}. \qquad \text{(A)}$$

1 Beispiele zu Sonderanweisungen finden sich in Bd. 2.
2 Heiner Müller-Merbach. Operations Research: Methoden und Modelle der Optimalplanung. 3. Aufl., München 1973, S. 444–449.

Zur Prognose unter Annahme eines linearen Trends

$$y_{t+dt} = a_t + b_t dt$$

wobei a_t den Grundwert und b_t die Zeitsteigung im Zeitpunkt t darstellen, wird angenommen, dass alle Messwerte y_{t-1} auf der Trendgeraden liegen und der Gleichung

$$y_{t-i} = a_t - b_t i$$

genügen. Zufallsschwankungen werden vernachlässigt. Damit lässt sich die Zeitsteigung nach

$$b_t = y_t^{(2)} - y_{t-1}^{(2)} \qquad \text{(B)}$$

aus der Differenz der Mittelwerte 2. Ordnung zweier aufeinanderfolgender Perioden berechnen. Für den Grundwert a_t gilt eine ebenso einfache Beziehung

$$a_t = 2y_t^{(1)} - y_t^{(2)}. \qquad \text{(C)}$$

Ein Prognosewert y^p_{t+1} für die Periode t + 1 ergibt sich dann als lineare Fortsetzung der bisherigen Zeitreihe nach

$$y^p_{t+1} = a_t + b_t \qquad \text{(D)}$$

da dt gleich 1 ist.
Bei der praktischen Anwendung dieser Methode der exponentiellen Glättung 2. Ordnung ergibt sich das Problem, zunächst Ausgangswerte für a_t und b_t aus Anfangswerten der Messreihe zu ermitteln und eine Startperiode für die Glättung zu definieren. Hierzu werden in dem folgenden Programmbeispiel die Startwerte a_s und b_s aus einer vorgebbaren Anzahl von Anfangsmesswerten mittels der linearen Trendrechnung[3] nach den Beziehungen

$$a_s = \frac{1}{M} \sum_{i=1}^{M} y_i$$

und

$$b_s = \frac{12}{M(M^2-1)} \sum_{i=1}^{M} y_i \left(i - \frac{M+1}{2}\right)$$

3 Müller-Merbach, Operations Research, S. 443–444.

ermittelt. Aus den Formeln B und C lassen sich die dazugehörigen Mittelwerte der Startperiode s errechnen.

$$y_s^{(1)} = a_s - (1-\alpha)b_s/\alpha$$

$$y_s^{(2)} = a_s - 2(1-\alpha)b_s/\alpha$$

Die Werte der Perioden für t>s ergeben sich aus der Anwendung der Gleichung A für die Mittelwerte, der Gleichungen B und C für den Grundwert a_t resp. für die Zeitsteigung b_t und der Gleichung D für den Prognosewert y^P_{t+1}.
Die Bedeutung der Programmvariablen ist im Programm selbst durch Kommentare erläutert.

```
  10  REM  EXPONENTTELLE GLAETTUNG ZWEITER ORDNUNG
  21  COMMENT Y(T,1) = MESSWERTE
  22  COMMENT Y(T,2) = MITTELWERT 1.ORDNUNG
  23  COMMENT Y(T,3) = MITTELWERT 2.ORDNUNG
  24  COMMENT Y(T,4) = PROGNOSEWERT
  25  COMMENT A(T) = GRUNDWERT
  26  COMMENT B(T) = ZEITSTEIGUNG
  30  PRINT ≠ARBEITSTABELLE≠
  35  READ N,Z,M,S
  40  PRINT ≠***************≠
  41  PRINT ≠GLAETTUNGSKONSTANTE≠,≠              Z =≠,Z
  42  PRINT ≠PERIODENANZAHL≠,≠ ≠,≠             N =≠,N
  43  PRINT ≠ANZAHL DER STARTWERTE≠,≠              M =≠,M
  44  PRINT ≠STARTPERIODE DER GLAETTUNG≠,≠                S =≠,S
  50  PRINT  ≠   ZAHL DER≠,≠EFFEKTIVER≠,≠MITTELWERT 1.≠,≠MITTELWERT 2.≠,
  55  PRINT  ≠PROGNOSEWERT≠
  60  PRINT  ≠   PERIODEN≠,≠ABSATZ IN≠,≠ORDNUNG IN T≠,≠ORDNUNG IN T≠,≠  FUER DIE≠
  65  PRINT ≠         ≠,≠PERIODE T≠,≠          ≠,≠          ≠,≠FOLGEPERIODE≠,
  67  PRINT ≠  GRUNDWERT≠,≠  ZEITSTEIGUNG≠
  70  PRINT ≠        T≠,≠     Y(T)≠,≠      Y(T,2)≠,≠      Y(T,3)≠,≠      Y(T,4)  ≠,
  75  PRINT ≠        A(T)≠,≠        B(T)≠
  90  PRINT
  100 PRINT
  110 DIM Y(30,4),A(30),B(30)
  115 FOR T=1 TO N
  120   READ Y(T,1)
  125 NEXT T
  130 FOR T=1 TO M
  135   LET A(S)=A(S)+Y(T,1)
  140   LET B(S)=B(S)+Y(T,1)*(T-(M+1)/2)
  145   IF T >= S GOTO 155
  150   PRINT T,Y(T,1)
  155 NEXT T
  156 LET A(S)=A(S)/M
  157 LET B(S)=B(S)*12/(M*(M**2))
  158 FOR T=S TO N
  159   IF T > S GOTO 165
  160   LET Y(T,2)=A(T)-((1-Z)/Z)*B(T)
  161   LET Y(T,3)=A(T)-(2*(1-Z)/Z)*B(T)
  162   LET Y(T,4)=A(T)+B(T)
  163   GOTO 210
  165   LET Y(T,2)=Z*Y(T,1)+(1-Z)*Y(T-1,2)
  170   LET Y(T,3)=Z*Y(T,2)+(1-Z)*Y(T-1,3)
  190   LET A(T)=2*Y(T,2)-Y(T,3)
  200   LET B(T)=Y(T,3)-Y(T-1,3)
  202   Y(T,4)=A(T)+B(T)
  204   IF T <> S+1 GOTO 210
  206   PRINT ≠START DER GLAETTUNG≠
  210   PRINT T,Y(T,1),Y(T,2),Y(T,3),Y(T,4),A(T),B(T)
  220 NEXT T
  455 DATA 27,0,2,5,3
  458 DATA 192,201,198,203,206,208,215,209,214,220,225,232,235,241,244,248,251
  459 DATA 255,259,266,263,265,268,274,278,279,283
  460 END

*** EVERYTHING SEEMS OK -- LET≠S GO AHEAD

   PERCENT OF AVAILABLE STORAGE USED                  37.892
   PERCENT OF AVAILABLE DATA STORAGE USED              9.394
   PERCENT OF AVAILABLE NUMBERED STATEMENTS USED      14.118
```

ARBEITSTABELLE
****

GLAETTUNGSKONSTANTE Z = 0.2
PERIODENANZAHL N = 27
ANZAHL DER STARTWERTE M = 5
STARTPERIODE DER GLAETTUNG S = 3

ZAHL DER PERIODEN T	EFFEKTIVER ABSATZ IN PERIODE T Y(T)	MITTELWERT 1. ORDNUNG IN T Y(T,2)	MITTELWERT 2. ORDNUNG IN T Y(T,3)	PROGNOSEWERT FUER DIE FOLGEPERIODE Y(T,4)	GRUNDWERT A(T)	ZEITSTEIGUNG B(T)
1	192					
2	201					
3	198	188.48	176.96	282.88	200	2.88
START DER GLAETTUNG						
4	203	191.384	179.845	205.808	202.923	2.8848
5	206	194.307	182.737	208.77	205.877	2.89248
6	208	197.046	185.599	211.354	208.493	2.8617
7	215	200.637	188.607	215.674	212.667	3.00753
8	209	202.309	191.347	216.012	213.272	2.74056
9	214	204.647	194.007	217.948	215.288	2.66007
10	220	207.718	196.749	221.429	218.687	2.74216
11	225	211.174	199.634	225.599	222.714	2.88501
12	232	215.339	202.775	231.045	227.904	3.14104
13	235	219.272	206.075	235.768	232.469	3.29925
14	241	223.617	209.583	241.16	237.651	3.50854
15	244	227.694	213.205	245.805	242.182	3.62214
16	248	231.755	216.915	250.305	246.595	3.70996
17	251	235.604	220.653	254.293	250.555	3.73776
18	255	239.483	224.419	258.313	254.547	3.76605
19	259	243.387	228.213	262.354	258.561	3.79351
20	266	247.909	232.152	267.606	263.667	3.93934
21	263	250.927	235.907	269.703	265.948	3.7551
22	265	253.742	239.474	271.577	268.01	3.56699
23	268	256.594	242.898	273.713	270.289	3.42391
24	274	260.075	246.333	277.252	273.816	3.43539
25	278	263.66	249.799	280.986	277.521	3.46532
26	279	266.728	253.184	283.657	280.271	3.38586
27	283	269.982	256.544	286.78	283.421	3.35957

Die Critical Path Method gehört zu den meistverwendeten Methoden der Netzplantechnik[4]. Von der Aufgabenstellung her betrachtet, kann CPM primär zur Struktur-, Zeit- und Kostenanalyse eingesetzt werden.[5]
Aufgabe der Strukturanalyse ist es, die technologische Struktur eines Projektes zu analysieren und den Projektablauf graphisch und/oder tabellarisch so darzustellen, dass die Aktivitäten und deren Abhängigkeiten klar ersichtlich werden (Vorgangorientierter Netzplan).
Beginn und Ende jeder Aktivität werden durch spezifische Ereignisse angezeigt. Diese werden bei graphischer Darstellung durch Knoten, die Aktivitäten dagegen durch Pfeile (Kanten) dargestellt (Vorgangspfeil-Netzplan).
Jeder CPM-Netzplan muss so gestaltet sein, dass er nur eine Quelle (Startereignis des Projektes) und eine Senke (Endereignis des Projektes) enthält und im übrigen zusammenhängend und zyklusfrei ist. Weiterhin ist zu beachten, dass zwei Aktivitäten nicht gemeinsame Anfangs- und Endereignisse aufweisen dürfen, was durch Einfügen von Scheinaktivitäten verhindert werden kann.
Die Zeitanalyse vollzieht sich in zwei Phasen. Zunächst wird für jede Aktivität (i,j) die Zeitdauer (d_{ij}) bestimmt; für Scheinaktivitäten gilt $d_{ij}=0$. Es wird nur mit einem Zeitwert pro Aktivität gearbeitet (Einzeit-Verfahren).
Die zweite Phase der Zeitanalyse dient zur Ermittlung des sogenannten kritischen Weges und der Pufferzeiten.
Falls in einem insgesamt n Ereignisse umfassenden Projekt P der Startzeitpunkt auf

$$t_1^{(0)} = 0$$

festgelegt und der frühestmögliche Abschluss einer Aktivität $(i,j) \in P$ mit $t_j^{(0)}$ bezeichnet wird, gilt in der Vorwärtsrechnung für alle Aktivitäten, dass

$$t_j^{(0)} = \max(t_i^{(0)} + d_{ij})$$

mit $i < j$ und $2 \leqq j \leqq n$ ist.
Die frühestmögliche Abschlusszeit des Projektes P ist damit $t_n^{(0)}$.

4 Adolf Angermann. Netzwerke und Netzplantechnik. In: Erwin Grochla und Waldemar Wittman. Handwörterbuch der Betriebswirtschaft. 4. Aufl., Bd. 2, Stuttgart 1975, Sp. 2743–2776. Rolf Berg u. a. Netzplantechnik. Grundlagen, Methoden, Praxis. Zürich 1973.
5 Karl Weber. Projektanalyse mit CPM. Industrielle Organisation 37 (1968) 4, S. 236–248.

Wenn weiterhin der spätestmögliche Abschluss einer Aktivität (i, j) $\in P$ mit $t_i^{(1)}$ bezeichnet und $t_n^{(1)} = t_n^{(0)}$ gesetzt wird, gilt für alle übrigen Ereignisse des Projektes P aufgrund der Rückwärtsrechnung

$$t_i^{(1)} = \min(t_j^{(1)} - d_{ij}), i < j, 1 \leqq i \leqq n-1.$$

Anhand dieser Ereigniszeitpunkte kann der kritische Weg bestimmt werden. Er verläuft über jene Ereignisse, für die

$$t_i^{(0)} = t_i^{(1)},$$

$$t_j^{(0)} = t_j^{(1)}$$

und

$$t_j^{(1)} - t_i^{(0)} = d_{ij}$$

zutrifft.

Der kritische Weg stellt den zeitlich längsten Pfad vom Start- zum Endereignis eines Projektes dar; er beinhaltet ausschliesslich pufferzeitlose Aktivitäten.
Ist die zur Ausführung einer Aktivität maximal verfügbare Zeit grösser als die Aktivitätsdauer, ist also

$$t_j^{(1)} - t_i^{(0)} - d_{ij} > 0,$$

so ist diese Aktivität nicht kritisch. Diesfalls sind für die Aktivität (i, j) folgende Zeitpunkte wesentlich:

frühester Anfangszeitpunkt	$t_{ij}^{fa} = t_i^{(0)}$
spätester Anfangszeitpunkt	$t_{ij}^{sa} = t_j^{(1)} - d_{ij}$
frühester Endzeitpunkt	$t_{ij}^{fe} = t_i^{(0)} + d_{ij}$
spätester Endzeitpunkt	$t_{ij}^{se} = t_j^{(1)}$

Nichtkritische Aktivitäten verfügen demnach über Pufferzeiten, von denen in der Fachliteratur insbesondere vier diskutiert werden:

gesamte Pufferzeit	$t_{ij}^{tp} = t_j^{(1)} - t_i^{(0)} - d_{ij}$
freie Pufferzeit	$t_{ij}^{fp} = t_j^{(0)} - t_i^{(0)} - d_{ij}$
unabhängige Pufferzeit	$t_{ij}^{up} = \max(0, t_j^{(0)} - t_i^{(1)} - d_{ij})$

bedingte Pufferzeit $t_{ij}^{bp} = t_j^{(1)} - t_j^{(0)} = t_{ij}^{tp} - t_{ij}^{fp}$

Die gesamte Pufferzeit einer Aktivität $(i,j) \in P$ gibt den Zeitraum an, um den diese Aktivität verschoben oder die Aktivitätsdauer verlängert werden kann, ohne dass dadurch der Projektendtermin überschritten wird. Die freie Pufferzeit zeigt an, um welche Zeitspanne eine Aktivität ohne Einfluss auf die Gesamtprojektdauer verschoben oder ausgedehnt werden kann, wenn alle vorausgehenden Aktivitäten zum frühestmöglichen Termin beendet werden und für alle sequentiell folgenden Aktivitäten der frühestmögliche Beginn angesetzt wird.
Die unabhängige Pufferzeit läßt erkennen, um welche Zeitgrösse eine Aktivität (i, j) – unabhängig von der zeitlichen Stellung der vorausgehenden und nachfolgenden Aktivitäten – verschoben oder ausgedehnt werden kann, falls $t_i^{(1)}$ und $t_j^{(0)}$ planmässig eingehalten werden und

$$d_{ij} \leq t_j^{(0)} - t_i^{(1)}$$

ist.
Die bedingte Pufferzeit gibt an, um welche Zeitspanne der Endzeitpunkt einer Aktivität durch grösstmögliche Rückverlegung des Startzeitpunktes aller Folgeaktivitäten verschoben werden kann. Diese Zeitspanne entspricht der Differenz zwischen gesamter und freier Pufferzeit.
Die Berechnung der vorerwähnten Ereignis- und Pufferzeiten kann mit Hilfe des hier abgedruckten BASIC-Programms erfolgen, wobei die Ereignis-(Knoten-)Numerierung mit $i = 1$ beginnend lückenlos so zu erfolgen hat, dass für alle Aktivitäten (i,j) die Bedingung $i<j$ eingehalten wird. Die nach dieser Numerierungsregel bezeichneten Aktivitäten $(i,j) \in P$ können in beliebiger Folge über DATA-Statements eingegeben werden; sie werden programmintern in steigender Reihenfolge (Vektor A) geordnet und so strukturiert weiterverarbeitet.
Die Bedeutung der einzelnen Variablen wird im BASIC-Programm durch COMMENT-Statements erläutert. Das bearbeitete Zahlenbeispiel stammt aus der für CPM immer noch grundlegenden Studie von J. E. Kelley, Jr.[6]; es zeigt, dass durch computergestützte Datenverarbeitung Fehler, wie sie bei manueller Rechnung häufig – hier auch bei Kelley – vorkommen, vermieden werden.

```
1   COMMENT I = STAREREIGNIS
2   COMMENT J = ENDEREIGNIS
3   COMMENT D = ZEITDAUER DER AKTIVITAET VON I NACH J
4   COMMENT T = FRUEHESTER START DER AKTIVITAET VON I NACH J
5   COMMENT A2 = SPAETESTER START DER AKTIVITAET VON I NACH J
```

6 James E. Kelley, Jr. Critical-Path Planning and Scheduling: Mathematical Basis. Operations Research 9 (1961) 3, May–June, p. 296–320, hier p. 318.

```
6    COMMENT A3 = FRUHESTES ENDE DER AKTIVITAET VON I NACH J
7    COMMENT R = SPAETESTE ENDE DER AKTIVITAET VON I NACH J
8    COMMENT A4 = TOTALE PUFFERZEIT
9    COMMENT A5 = FREIE PUFFERZEIT
10   COMMENT A6 = UNABHAENGIGE PUFFERZEIT
11   COMMENT A7 = BEDINGTE PUFFERZEIT
12   DIM I(50),J(50),D(50),A(50),T(50),R(50)
20   REM I9=MAXIMALE AKTIVITAETSZAHL = DIM-GRENZE
30   LET I9=50
40   LET K=0
50   LET K=K+1
60   READ I(K),J(K),D(K)
65   IF I$(K) = ≠ENDE≠ GOTO 940
70   IF I(K) < 0 THEN 100
80   IF K >= I9 THEN 120
90   GOTO 50
100  LET N=K-1
110  GOTO 140
120  PRINT ≠UEBERSPEICHERUNG≠,I9,≠AKTIVITAETEN ERREICHT≠
130  STOP
140  REM KLASSIFIZIERUNG DER AKTIVITAETEN
200  FOR K=1 TO N
210    LET A(K)=K
220  NEXT K
230  FOR K2=1 TO N
240    LET K3=0
250    FOR K=1 TO N-K2
260      LET A1=A(K)
270      LET A2=A(K+1)
280      IF I(A1)<I(A2) GOTO 340
290      IF T(A1)>T(A2) GOTO 310
300      IF J(A1)<J(A2) GOTO 340
310      LET K3=K3+1
320      LET A(K)=A2
330      LET A(K+1)=A1
340    NEXT K
350    IF K3=0 GOTO 380
360  NEXT K2
370  REM INITIALISIERUNG
380  FOR K=1 TO N
390    LET T(K)=0
400    LET R(K)=0
410  NEXT K
420  REM VORWAERTSRECHNUNG
430  FOR K=1 TO N
440    LET I1=I(A(K))
450    LET J1=J(A(K))
460    LET M=T(I1)+D(A(K))
470    IF T(J1)=0 THEN 490
480    IF T(J1)>=M GOTO 500
490    LET T(J1)=M
500  NEXT K
510  REM RUECKWAERTSRECHNUNG
520  LET R(J(A(N)))=T(J(A(N)))
530  FOR K=N TO 1 STEP -1
540    LET I1=I(A(K))
550    LET M=R(J(A(K)))-D(A(K))
560    IF R(I1)=0 GOTO 580
570    IF R(I1)<=M GOTO 590
580    LET R(I1)=M
590  NEXT K
600  REM ERGEBNISSE
610  PRINT
620  PRINT ≠ AKTIVITAET      START≠,
630  PRINT ≠     ENDE            PUFFERZEIT≠
640  PRINT ≠ I   J DAUER  FRUEH  SPAET≠,
650  PRINT ≠FRUEH SPAET   TOTAL  FREI  UNABH.  BEDINGT≠
660  PRINT
670  FOR K=1 TO N
680    LET A1=A(K)
690    LET I1=I(A1)
700    LET J1=J(A1)
701    LET A2=R(J1)-D(A1)
702    LET A3=T(I1)+D(A1)
703    LET A4=R(J1)-T(I1)-D(A1)
704    LET A5=T(J1)-T(I1)-D(A1)
705    LET A6=T(J1)-R(I1)-D(A1)
706    IF A6 >= 0 GOTO 708
707    LET A6=0
708    LET A7=A4-A5
710    PRINT TAB(1)↑I1↑TAB(5)↑J1↑TAB(9)↑D(A1)↑
720    PRINT TAB(15)↑T(I1)↑TAB(22)↑A2↑TAB(32)↑A3↑TAB(39)↑P(J1)↑
730    PRINT TAB(45)↑A4↑TAB(52)↑A5↑TAB(59)↑A6↑TAB(66)↑A7↑ ≠↑
```

```
770   IF R(J1)-T(T1)<>D(A1) GOTO 800
780   PRINT #KRITISCH#
790   GOTO 810
800   PRINT
810 NEXT K
812 PRINT
815 PRINT #PROJEKTDAUER =#,R(J(A(N)))
820 DATA 6,8,0,6,9,8,7,8,40,8,9,32,9,10,8,10,11,8,10,12,24,11,12,0,11,13,6
830 DATA 1,2,144,1,4,360,2,3,8,3,4,0,3,5,12,3,6,225,3,7,200,4,5,8,5,6,35
840 DATA 12,13,4,13,14,4,-1,-1,-1
845 DATA #ENDE#,-1,-1
850 FOR K=1 TO I9
860   LET I(K)=0
870   LET J(K)=0
880   LET D(K)=0
890   LET A(K)=0
900   LET T(K)=0
910   LET R(K)=0
920 NEXT K
930 GOTO 40
940 STOP
950 END
```

```
*** EVERYTHING SEEMS OK -- LET#S GO AHEAD

    PERCENT OF AVAILABLE STORAGE USED                47.081
    PERCENT OF AVAILABLE DATA STORAGE USED           20.000
    PERCENT OF AVAILABLE NUMBERED STATEMENTS USED    29.118
```

AKTIVITAET			START		ENDE			PUFFERZEIT			
I	J	DAUER	FRUEH	SPAET	FRUEH	SPAET	TOTAL	FREI	UNABH.	BEDINGT	
1	2	144	0.0	11	144	155	11	0.0	0.0	11	
1	4	360	0.0	0.0	360	360	0.0	0.0	0.0	0.0	KRITISCH
2	3	8	144	155	152	163	11	0.0	0.0	11	
3	4	0.0	152	360	152	360	208	208	197	0.0	
3	5	12	152	356	164	368	204	204	193	0.0	
3	6	225	152	178	377	403	26	26	15	0.0	
3	7	200	152	163	352	363	11	0.0	0.0	11	
4	5	8	360	360	368	368	0.0	0.0	0.0	0.0	KRITISCH
5	6	35	368	368	403	403	0.0	0.0	0.0	0.0	KRITISCH
6	8	0.0	403	403	403	403	0.0	0.0	0.0	0.0	KRITISCH
6	9	8	403	427	411	435	24	24	24	0.0	
7	8	40	352	363	392	403	11	11	0.0	0.0	
8	9	32	403	403	435	435	0.0	0.0	0.0	0.0	KRITISCH
9	10	8	435	435	443	443	0.0	0.0	0.0	0.0	KRITISCH
10	11	8	443	457	451	465	14	0.0	0.0	14	
10	12	24	443	443	467	467	0.0	0.0	0.0	0.0	KRITISCH
11	12	0.0	451	467	451	467	16	16	2	0.0	
11	13	6	451	465	457	471	14	14	0.0	0.0	
12	13	4	467	467	471	471	0.0	0.0	0.0	0.0	KRITISCH
13	14	4	471	471	475	475	0.0	0.0	0.0	0.0	KRITISCH

```
PROJEKTDAUER =        475
```

42 Volkswirtschaftslehre

421 Preiselastizität der Nachfrage

Unter der direkten Preiselastizität versteht man das Verhältnis zwischen der relativen Änderung der mengenmässigen Nachfrage nach einem Gut und der sie bewirkenden relativen Änderung des Preises dieses Gutes[7].

7 Erich Schneider. Einführung in die Volkswirtschaftstheorie. Wirtschaftspläne und wirtschaftliches Gleichgewicht in der Verkehrswirtschaft. 11. Aufl., Tübingen 1967, S. 31. Vgl. auch: Alfred E. Ott. Grundzüge der Preistheorie. 2. Aufl., Göttingen 1970, S. 134–139.

Ausgehend von einer Nachfragefunktion $m = f(p)$ bestimmt sich die direkte Preiselastizität der Nachfrage $E_{m,p}$ nach der Formel

$$E_{m,p} = \frac{\frac{dm}{m}}{\frac{dp}{p}},$$

wobei m die nachgefragte Menge und p den dazugehörigen Preis darstellen. Eine Abhängigkeit von weiteren Variablen – weiteres Gut oder Volkseinkommen – wird hier nicht unterstellt.
Da nach dieser Definition die Elastizität normalerweise negativ wäre, wird nach einem Vorschlag von Alfred Marshall die Preiselastizität mit einem Minuszeichen versehen und in der Form

$$E_{m,p} = -\frac{dm}{m}\,\frac{p}{dp} = -\frac{p}{m}\,\frac{dm}{dp}$$

geschrieben. Der Wert von dm/dp kann nur berechnet werden, wenn die Nachfragefunktion explizit angegeben ist; ansonsten muss er aus zwei Wertepaaren m_1, p_1 und m_2, p_2 näherungsweise nach

$$\frac{\Delta m}{\Delta p} = \frac{m_1 - m_2}{p_1 - p_2}$$

bestimmt werden.
In der Literatur wird zwischen Punkt- und Bogenelastizität unterschieden.
Zur praktischen Berechnung der Punktelastizität verwendet man die Beziehung

$$E^{(A)}_{m,p} = -\frac{m_1}{p_1}\,\frac{m_1 - m_2}{p_1 - p_2}$$

oder, falls m_2 und p_2 als Bezugspunkte dienen[8],

$$E^{(B)}_{m,p} = -\frac{p_2}{m_2}\,\frac{m_1 - m_2}{p_1 - p_2}.$$

Zur Berechnung der Bogenelastizität werden in der Literatur fünf Formeln diskutiert[9]:

8 Helga Luckenbach. Theorie des Haushalts. Göttingen 1974, S. 67–76.
9 Clark L. Allen. The Framework of Price Theory. Belmont 1967, p. 68–71.

1. Preis- und Mengenänderungen beziehen sich auf Durchschnittspreis und Durchschnittsmenge.

$$E_{m,p}^{(C)} = -\frac{p_1 + p_2}{m_1 + m_2}\,\frac{m_1 - m_2}{p_1 - p_2}$$

2. Die Preisänderung bezieht sich auf den neuen Preis, während sich die Mengenänderung auf die alte Menge bezieht.

$$E_{m,p}^{(D)} = -\frac{p_2}{m_1}\,\frac{m_1 - m_2}{p_1 - p_2}$$

3. Die Preisänderung bezieht sich auf den alten Preis; die Mengenänderung auf die neue Menge.

$$E_{m,p}^{(E)} = -\frac{p_1}{m_2}\,\frac{m_1 - m_2}{p_1 - p_2}$$

4. Werden die Änderungen auf die neuen Werte bezogen, so erhält man die Formel nach $E_{m,p}^{(B)}$.

5. Werden Preis- und Mengenänderungen in Relation zu den alten Werten gesetzt, so ergibt sich die Gleichung nach $E_{m,p}^{(A)}$.

Die Bogenelastizität lässt sich auch auf logarithmischer Basis nach einer ebenso leistungsfähigen Formel berechnen[10]

$$E_{m,p}^{(F)} = -\frac{\lg(m_1) - \lg(m_2)}{\lg(p_1) - \lg(p_2)}$$

Das folgende BASIC-Programm ermittelt die Preiselastizitäten der Nachfrage nach den oben angegebenen Gleichungen A bis F.

```
PRINT *PREISELASTIZITAET DER NACHFRAGE*
COMMENT P = PREIS                M = MENGE
COMMENT A = PUNKTELASTIZITAET-1 BEZOGEN AUF I-1
COMMENT B = PUNKTELASTIZITAET-2 BEZOGEN AUF I
COMMENT C = BOGENELASTIZITAET-1 DURCHSCHNITTSWERTBASIS
COMMENT D = BOGENELASTIZITAET-2 PREISAENDERUNG BEZOGEN AUF NEUEN PREIS,
COMMENT     MENGENAENDERUNG BEZOGEN AUF AUSGANGSMENGE
COMMENT E = BOGENELASTIZITAET-3 PREISAENDERUNG BEZOGEN AUF AUSGANGSPREIS,
COMMENT     MENGENAENDERUNG BEZOGEN AUF NEUE MENGE
COMMENT F = BOGENELASTIZITAET AUF LOGERITHMISCHER 21S9S
COMMENT DIMENSIONIERUNG DER VEKTOREN
DIM P(100),M(100)
```

10 Heinrich v. Stackelberg. Grundlagen der theoretischen Volkswirtschaftslehre. 2. Aufl., Tübingen 1951, S. 179. Rudolf Richter. Preistheorie. Wiesbaden. 1963, S. 115–117.

```
    COMMENT N = ZAHL DER EINGELESENEN WERTE
 10 READ N
    IF N$=≠ENDE≠ GOTO 20
    PRINT
    PRINT
    PRINT ≠      PREIS≠,≠      MENGE≠,≠      PUNKT-≠,≠      PUNKT-≠,≠      BOGEN-≠,
    PRINT ≠       BOGEN-≠,≠       BOGEN-≠,≠       BOGEN-≠
    PRINT ≠ ≠,≠ ≠,≠ ELASTIZITAET-1≠,≠ ELASTIZITAET-2≠,≠ ELASTIZITAET-1≠,
    PRINT ≠ ELASTIZITAET-2≠,≠ ELASTIZITAET-3≠,≠ ELASTIZITAET-4≠
    PRINT
    COMMENT EINLESEN DER P- UND M-WERTE
    FOR I = 1 TO N
      READ P(I),M(I)
    NEXT I
    COMMENT ELASTIZITAETSBERECHNUNG
    PRINT P(1),M(1),≠AUSGANGSPUNKT≠
    FOR I = 2 TO N
      LET A=(M(I-1)-M(I))/M(I-1)*P(I-1)/(P(I-1)-P(I))
      LET B=(M(I-1)-M(I))/M(I)*P(I)/(P(I-1)-P(I))
      LET C=(M(I-1)-M(I))/(M(I-1)+M(I))*(P(I-1)+P(I))/(P(I-1)-P(I))
      LET D=P(I)/M(I-1)*(M(I-1)-M(I))/(P(I-1)-P(I))
      LET E=P(I-1)/M(I)*(M(I-1)-M(I))/(P(I-1)-P(I))
      LET F=(LOG(M(I))-LOG(M(I-1)))/(LOG(P(I))-LOG(P(I-1)))
      COMMENT DRUCK,PREISELASTIZITAET NEGATIV DEFINIERT
      PRINT P(I),M(I),-A,-B,-C,-D,-E,-F
    NEXT I
    GO TO 10
 20 STOP
    END LIST DATA


LIST OF DATA AFTER END STATEMENT


10,134,1,124,2,114,3,104,4,94,5,84,6,74,7,64,8,54,9,44,10
2,12,32,10,40
≠ENDE≠
*

*** EVERYTHING SEEMS OK -- LET≠S GO AHEAD

  PERCENT OF AVAILABLE STORAGE USED                29.270
  PERCENT OF AVAILABLE DATA STORAGE USED            0.800
  PERCENT OF AVAILABLE NUMBERED STATEMENTS USED     0.588
```

PREISELASTIZITAET DER NACHFRAGE

PREIS	MENGE	PUNKT-ELASTIZITAET-1	PUNKT-ELASTIZITAET-2	BOGEN-ELASTIZITAET-1	BOGEN-ELASTIZITAET-2	BOGEN-ELASTIZITAET-3	BOGEN-ELASTIZITAET-4
134	1	AUSGANGSPUNKT					
124	2	13.4	6.2	8.6	12.4	6.7	8.93712
114	3	6.2	3.8	4.76	5.7	4.13333	4.82219
104	4	3.8	2.6	3.11429	3.46667	2.85	3.13353
94	5	2.6	1.88	2.2	2.35	2.08	2.20724
84	6	1.88	1.4	1.61818	1.68	1.56667	1.62095
74	7	1.4	1.05714	1.21538	1.23333	1.2	1.21616
64	8	1.05714	0.8	0.92	0.914286	0.925	0.919752
54	9	0.8	0.6	0.694118	0.675	0.711111	0.693253
44	10	0.6	0.44	0.515789	0.488889	0.54	0.51447

PREIS	MENGE	PUNKT-ELASTIZITAET-1	PUNKT-ELASTIZITAET-2	BOGEN-ELASTIZITAET-1	BOGEN-ELASTIZITAET-2	BOGEN-ELASTIZITAET-3	BOGEN-ELASTIZITAET-4
12	32	AUSGANGSPUNKT					
10	40	1.5	1	1.22222	1.25	1.2	1.2239

In seinem Buch „A Contribution to the Theory of the Trade Cycle" (1950) hat J. R. Hicks ein gesamtwirtschaftliches Oszillationsmodell geschaffen, das zur Untersuchung konjunktureller Schwankungen um einen ansteigenden Trend dient und im wesentlichen auf einfachen Annahmen, deren empirische Relevanz hier nicht zur Diskussion steht, basiert[11].
Nach Hicks steigen die Investitionen des Staates unabhängig von der konjunkturellen Entwicklung in den Perioden $t = 0, 1, 2, 3, \ldots, n$ kontinuierlich an. Für diese autonomen Investitionen gilt

$$I_t^{aut} = A_o(1 + r)^t,$$

wobei A_0 die Basisinvestition des Staates in der Periode $t = 0$ und r eine konstante positive Wachstumrate der autonomen Investitionen darstellen.
Die induzierten Investitionen werden durch das Wachstum der Wirtschaft, das nach dem Akzeleratorprinzip an der Veränderung des Volkseinkommens gemessen wird, bestimmt. Nach Hicks gilt für diese induzierten Investitionen

$$I_t^{ind} = v(Y_{t-1} - Y_{t-2}),$$

wobei Y_{t-1} das Volkseinkommen der Periode t-1 und Y_{t-2} entsprechend das Volkseinkommen der Periode t-2 darstellen. v steht für den konstanten positiven Akzeleratorwert.
Die induzierten Investitionen sind zweiseitig begrenzt. Die obere Grenze wird durch die maximale Produktionskapazität im Investitionssektor der Volkswirtschaft – analog zu den autonomen Investitionen – durch

$$I_t^{ind,max} = A_3(1 + r_3)^t$$

bestimmt, wobei A_3 den Bestand an Produktionsanlagen in der Periode $t = 0$ und r_3 eine für alle Perioden gleichbleibende Wachstumsrate darstellen.
Als untere Grenze für die induzierten Investitionen wird

$$I_t^{ind,min} = A_4 \cdot K_{t-1}$$

11 Vgl. Walter A. Jöhr. Die Konjunkturschwankungen. Tübingen 1952, S. 558–573; Jürgen Kromphardt. Wachstum und Konjunktur. Grundlagen einer theoretischen Analyse und wirtschaftspolitischen Steuerung. Göttingen 1972, S. 169–180.

angesetzt. Dieser Wert zeigt den – bei Ausfall aller (Neu- und) Ersatzinvestitionen – maximalen Investitionsrückgang an, wobei A_4 als negative Investitionsquote zu interpretieren ist und K_{t-1} für den Bestand an Produktionsanlagen in der vorausgehenden Periode t-1 steht.
Die Gesamtinvestitionen I_t ergeben sich als Summe der autonomen und induzierten Investitionen, so dass

$$I_t = I_t^{aut} + I_t^{ind}$$

gilt.
Nach Hicks ist der Konsum vom Volkseinkommen – und damit implizit von der Gesamtproduktion – der vorhergehenden Periode abhängig. Falls c die konstant bleibende Konsumquote darstellt, ergibt sich daraus die Gleichung

$$C_t = c \cdot Y_{t-1}.$$

Auch der Konsum ist mit

$$C_t^{max} = A_1 (1 + r_1)^t$$

und

$$C_t^{min} = A_2 (1 + r_2)^t$$

plafoniert, wobei über den Mindestkonsum C_t^{min} ein negatives Sozialprodukt ausgeschlossen und C_t^{max} durch die maximale Produktionskapazität im Konsumsektor verursacht wird.
Das Volkseinkommen einer Periode entspricht generell den Gesamtinvestitionen und dem Konsum, so dass

$$Y_t = I_t + C_t$$

zutrifft.
Der Bestand an Produktionsanlagen setzt sich aus dem Kapitelbestand der Vorperiode, K_{t-1}, und den in der jeweiligen Periode induzierten Investitionen zusammen:

$$K_t = K_{t-1} + I_t^{ind}$$

Die Kenntnis von K_{t-1} ist zur Berechnung der Untergrenze der induzierten Investitionen in der Periode t erforderlich.
Bei der Programmierung des Hicks-Modells wurden die oben explizit angegebenen Gleichungen verwendet, wobei auf die graphische Darstellung oszillierender Kenngrössen besonderer Wert gelegt wurde. Dem entspre-

chend wurde das Hicks-Modell als Unterprogramm in ein BASIC-Hauptprogramm integriert, das Wertepaare aus beliebigen Funktionen oder Modellen aufzuzeichnen vermag.
Bezüglich der Abarbeitung des Hicks-Modells im Unterprogramm ist zu bemerken, dass zur Berechnung der in Periode t anfallenden Basiswerte jeweils nur die Daten von zwei vorangehenden Perioden (t-2 und t-1) benötigt werden. Diese werden mit jedem Periodenübergang jeweils zeitverschoben, im Sinne von

$$Y_{t-2} \Leftarrow Y_{t-1}$$

$$Y_{t-1} \Leftarrow Y_{t}$$

und

$$K_{t-1} \Leftarrow K_{t}$$

umgespeichert und stehen damit soweit erforderlich für Neuberechnung in der nachfolgenden Periode t zur Verfügung.
Hinsichtlich der graphischen Auswertung der Kenngrössen im Hauptprogramm ist zu beachten, dass nach dem Einlesen der Modellparameter und des Auswertungsbereichs zunächst der grösste und kleinste Wert der aufzuzeichnenden Variablen bestimmt und hieraus, bezogen auf 75 Druckstellen, ein Skalierungsfaktor für das Y-Intervall berechnet wird. Anschliessend erfolgt das Aufzeichnen der Funktion. Dabei muss das Modell – da in der vorausgehenden Arbeitsphase die permanente Speicherung der Auswertungsergebnisse unterblieb – auf seine Anfangswerte zurückgesetzt und nochmals durchgerechnet werden. Falls das Y-Intervall den Wert Null mitenthält, wird auch die Nullinie gedruckt.
Die Bedeutung der Variablen und der Modellparameter wird im BASIC-Programm durch COMMENT-Statements ausführlich erläutert[12].

```
5     DIM I(3),J(3),K(3),Z(3),L(3),C(3),M$(3)
10    LET U$=# #
20    LET V$=#*#
30    LET Z$=#I#
31    REM EINLESEN DER MODELLPARAMETER
32    READ C1,V1,A0,R,A1,R1,A2,R2,A3,R3,A4,L(2),Z(1),Z(2),M9,M$(1),M$(2),M$(3)
33    DATA 0.75,3,250,0.03,1000,0.03,500,0.03,1000,0.03,-0.1,3000,1000,1000
34    DATA 3,#GES.INVESTITION#
35    COMMENT C1 = KONSUMQUOTE
36    COMMENT V1 = RATE DER INDUZIERTEN INVESTITIONEN
37    COMMENT A0 UND R FUNKTIONSPARAMETER DER AUTONOMEN INVESTITIONEN
38    COMMENT A1 UND R1 KONSUMPARAMETER DER OBERGRENZE
39    COMMENT A2 UND R2 KONSUMPARAMETER DER UNTERGRENZE
40    COMMENT A3 UND R3 INDUZIERTE INVESTITIONS PARAMETER DER OBERGRENZE
41    COMMENT A4 INDUZIERTE INVESTITIONS-PARAMETER DER UNTERGRENZE
```

12 Beispieldaten entnommen aus: Kromphardt, Wachstum und Konjunktur, S. 180.

```
42   COMMENT L(2) = ANFANGSKAPITALBESTAND VOR DER PERIODE T
43   COMMENT Y(2) UND Y(1) SIND VOLKSEINKOMMEN VOR DER PERIODE T
44   COMMENT N9 = STEUERPARAMETER ZUR AUSWAHL DER AUFZUZEICHNEMDEN VARIABLEN
45   COMMENT N9=1 ZEICHNET K(T)
46   COMMENT N9=2 ZEICHNET J(T)
47   COMMENT N9=3 ZEICHNET I(T)
48   COMMENT N9=4 ZEICHNET C(T)
49   COMMENT N9=5 ZEICHNET Z(T)
50   COMMENT N9=6 ZEICHNET L(T)
51   PRINT ≠            C1 =≠,C1,≠          V1 =≠,V1
52   PRINT ≠            A0 =≠,A0,≠           R =≠,R
53   PRINT ≠            A1 =≠,A1,≠          R1 =≠,R1
54   PRINT ≠            A2 =≠,A2,≠          R2 =≠,R2
55   PRINT ≠            A3 =≠,A3,≠          R3 =≠,R3
56   PRINT ≠            A4 =≠,A4,≠        L(2) =≠,L(2)
57   PRINT ≠          Z(1) =≠,Z(1),≠        Z(2) =≠,Z(2)
58   PRINT ≠            N9 =≠,N9
59   REM X1 = ANFANG
60   REM X2 = ANZAHL DER PERIODEN
61   READ X1,X2
65   DATA 0,50
70   LET X2=X1+X2
80   LET X3=1
95   PRINT ≠ ≠,≠    AUTONOME≠,≠   INDUZIERTE≠,≠    GESAMT-≠
96   PRINT ≠    PERIODE≠,≠ INVESTITIONEN≠,≠ INVESTITIONEN≠,≠ INVESTITIONEN≠,
97   PRINT ≠      KONSUM≠,≠VOLKSFINKOMMEN≠,≠KAPITALBESTAND≠
100  REM BESTIMMEN DES MINIMALEN UND MAXIMALEN Y-WERTES
102  LET I9=0
105  FOR X=X1 TO X2 STEP X3
109    GOSUB 1000
110    LET I9=I9+1
111    IF I9 > 1 GOTO 115
112    LET Y1=Y
113    LET Y2=Y
114    LET Y4=Y
115    IF Y GT Y1 THEN 130
120    IF Y LT Y2 THEN 140
125    GOTO 142
130    LET Y1=Y
135    GOTO 142
140    LET Y2=Y
142    PRINT  X,K(T),J(T),I(T),C(T),Z(T),L(T)
145  NEXT X
146  IF N9 LE 0 GOTO 790
147  IF N9 GT 6 GOTO 790
148  COMMENT FUER N9 < 1 UND N9 > 6 WIRD NICHT GEZEICHNET
151  FOR T=1 TO 3
152    LET C(T)=0
153    LET I(T)=0
154    LET J(T)=0
155    LET K(T)=0
156    LET L(T)=0
157    LET Z(T)=0
158  NEXT T
159  RESTORE
160  PAGE
161  READ C1,V1,A0,R,A1,R1,A2,R2,A3,R3,A4,L(2),Z(1),Z(2),N9
162  COMMENT Y4 = MAXIMALE ANZAHL DER ZEICHENPUNKTE IN Y-RICHTUNG
163  LET Y4=75
164  LET Y3=ABS((Y1-Y2)/Y4)
165  PRINT ≠STARTWERTE   X=≠,X1,≠              Y=≠,Y4
170  PRINT ≠ENDWERTE     X=≠,X,≠               Y=≠,Y
175  PRINT ≠  SCHRITTWEITE ≠,≠IN X-RICHTUNG =≠,X3
180  PRINT ≠  SCHRITTWEITE ≠,≠IN Y-RICHTUNG =≠,Y3
185  PRINT ≠MAXIMALER Y-WERT≠,Y1
190  PRINT ≠MINIMALER Y-WERT≠,Y2
195  PRINT
197  PRINT  ≠AUFGEZEICHNET WIRD *≠;M$(1);M$(2);M$(3);≠* ALS FUNKTION DER ZEIT≠
200  FOR I=1 TO Y4+20
210    PRINT USING 710,V$;
220  NEXT I
230  PRINT USING 710,V$
380  REM NULLPUNKT WENN IN ZEICHENEBENE BERECHNEN
390  LET T1=0
400  IF Y2 >= 0 GOTO 480
410  IF Y1 < 0 GOTO 480
420  LET T3=INT(ABS(Y2)/Y3+0.5)
480  FOR X=X1 TO X2 STEP X3
500    GOSUB 1000
530    LET T1=INT(ABS((Y-Y2)/Y3))+1
565    IF T1 LT T3 GOTO 570
566    LET T2=T1
567    GOTO 610
```

```
570     LET T2=T3
610     PRINT USING 620,X1Y1
620     Z###.#  #######.##Z
630     FOR I=1 TO T2
640       IF I NE T3 GOTO 670
650       PRINT USING 710,Z$1
660       GOTO 720
670       IF I <> T1 GOTO 700
680       PRINT USING 710,V$1
690       GOTO 720
700       PRINT USING 710,U$1
710       Z=X
720     NEXT I
730     PRINT USING 710,U$
740   NEXT X
750   STOP
1000  REM UNTERPROGRAMM DES OSZILLATIONSMODELLS NACH HICKS
761   REM K= AUTONOME INVESTITIONEN
762   LET T=3
763   LET K(T)=A0*(1+R)**X
764   REM J = INDUZIERTE INVESTITIONEN
765   LET  J(T)=V1*(Z(T-1)-Z(T-2))
766   IF J(T) < A3*(1+R3)**X GOTO 880        ▾ OBERGRENZE DER INDUZIERTEN INVEST.
767   LET J(T)=A3*(1+R3)**X
800   IF J(T) > A4*L(T-1) GOTO 810            ▾ UNTERGRENZE DER INDUZIERTEN INVEST.
805   LET J(T)=A4*L(T-1)
810   REM I = GESAMTINVESTITIONEN
813   LET I(T)=K(T)+J(T)
814   REM C = KONSUM
815   LET C(T)=C1*Z(T-1)
816   IF C(T) <= A1*(1+R1)**X GOTO 820        ▾ OBERGRENZE DES KONSUM
817   LET C(T)=A1*(1+R1)**X
818   GOTO 830
820   IF C(T) > A2*(1+R2)**X GOTO 830         ▾ UNTERGRENZE DES KONSUM
825   LET C(T)= A2*(1+R2)**X
830   REM Z = VOLKSEINKOMMEN
831   LET Z(T)=I(T)+C(T)
832   REM L = KAPITALBESTAND
833   LET L(T)=L(T-1)+J(T)
834   LET L(T-1)=L(T)
835   LET Z(T-2)=Z(T-1)
836   LET Z(T-1)=Z(T)
837   IF N9 <= 0 GOTO 900
838   IF N9 > 6 GOTO 900
839   GOTO 840,850,860,870,880,890 ON N9
840   LET Y=K(T)
845   GOTO 900
850   LET Y=J(T)
855   GOTO 900
860   LET Y=I(T)
865   GOTO 900
870   LET Y=C(T)
875   GOTO 900
880   LET Y=Z(T)
885   GOTO 900
890   LET Y=L(T)
900   RETURN
790   END
```

C1 =	0.75	V1 =	3
A0 =	250	R =	0.03
A1 =	1000	R1 =	0.03
A2 =	500	R2 =	0.03
A3 =	1000	R3 =	0.03
A4 =	-.1	L(2) =	3000
Z(1) =	1000	Z(2) =	1000
N9 =	3		

PERIODE	AUTONOME INVESTITIONEN	INDUZIERTE INVESTITIONEN	GESAMT-INVESTITIONEN	KONSUM	VOLKSEINKOMMEN	KAPITALBESTAND
0.0	250	0.0	250	750	1000	3000
1	257.5	0.0	257.5	750	1007.5	3000
2	265.225	22.5	287.725	755.625	1043.35	3022.5
3	273.182	107.55	380.732	782.513	1163.24	3130.05
4	281.377	359.683	641.06	872.433	1513.49	3489.73
5	289.819	1050.75	1340.57	1135.12	2475.69	4540.48
6	298.513	1194.05	1492.57	1194.05	2686.62	5734.53
7	307.468	632.798	940.266	1229.87	2170.14	6367.33
8	316.693	-636.733	-320.04	1266.77	946.73	5730.6
9	326.193	-573.06	-246.866	710.047	463.181	5157.54
10	335.979	-515.754	-179.775	671.958	492.184	4641.78
11	346.058	87.0082	433.067	692.117	1125.18	4728.79
12	356.44	1425.76	1782.2	843.888	2626.09	6154.55
13	367.133	1468.53	1835.67	1468.53	3304.2	7623.09
14	378.147	1512.59	1890.74	1512.59	3403.33	9135.68
15	389.492	297.378	686.87	1557.97	2244.84	9433.05
16	401.177	-943.305	-542.129	1604.71	1062.58	8489.75
17	413.212	-848.975	-435.763	826.424	390.661	7640.77
18	425.608	-764.077	-338.469	851.217	512.747	6876.7
19	438.377	366.26	804.636	876.753	1681.39	7242.96
20	451.528	1806.11	2257.64	1261.04	3518.68	9049.07
21	465.074	1860.29	2325.37	1860.29	4185.66	10909.4
22	479.026	1916.1	2395.13	1916.1	4311.23	12825.5
23	493.397	376.71	870.106	1973.59	2843.69	13202.2
24	508.199	-1320.22	-812.019	2032.79	1220.78	11882
25	523.444	-1188.2	-664.751	1046.89	382.138	10693.8
26	539.148	-1069.38	-530.228	1078.3	548.067	9624.39
27	555.322	497.789	1053.11	1110.64	2163.76	10122.2
28	571.982	2287.93	2859.91	1622.82	4482.73	12410.1

```
STARTWERTE    X=          0.0                    Y=        75
ENDWERTE      X=         51                      Y=    -1293.89
  SCHRITTWEITE IN X-RICHTUNG =          1
  SCHRITTWEITE IN Y-RICHTUNG =         89.66
MAXIMALER Y-WERT                     4807.1
MINIMALER Y-WERT                    -1917.4

AUFGEZEICHNET WIRD *GES.INVESTITION* ALS FUNKTION DER ZEIT
*****************************************************************************
  0.0       250.00                    I   *
  1.0       257.50                    I   *
  2.0       287.72                    I   *
  3.0       380.73                    I    *
  4.0       641.05                    I       *
  5.0      1340.56                    I            *
  6.0      1492.56                    I             *
  7.0       940.76                    I          *
  8.0      -320.04                  *  I
  9.0      -246.86                   * I
 10.0      -179.77                    *I
 11.0       433.06                    I     *
 12.0      1782.20                    I                 *
 13.0      1835.66                    I                 *
 14.0      1890.73                    I                  *
 15.0       686.86                    I       *
 16.0      -542.12                *   I
 17.0      -435.76                 *  I
 18.0      -338.46                  * I
 19.0       804.63                    I        *
 20.0      2257.63                    I                  *
 21.0      2325.36                    I                   *
 22.0      2395.12                    I                    *
 23.0       870.10                    I          *
 24.0      -812.01            *       I
 25.0      -664.75             *      I
 26.0      -530.22              *     I
 27.0      1053.11                    I             *
 28.0      2859.90                    I                        *
 29.0      2945.70                    I                         *
 30.0      3034.07                    I                          *
 31.0      1102.22                    I             *
 32.0     -1123.34         *          I
 33.0      -927.31          *         I
 34.0      -748.38           *        I
 35.0      1359.61                    I            *
 36.0      3622.84                    I                                *
 37.0      3731.53                    I                                 *
 38.0      3825.69                    I                                  *
 39.0      1342.91                    I            *
 40.0     -1487.49     *              I
 41.0     -1232.73       *            I
 42.0     -1000.26         *          I
 43.0      1739.73                    I                 *
 44.0      4589.31                    I                                    *
 45.0      4726.99                    I                                     *
 46.0      4807.10                    I                                      *
 47.0      1583.64                    I                *
 48.0     -1917.39*                   I
 49.0     -1591.35   *                I
 50.0     -1293.89      *             I
```

431 Statistische Masszahlen

Statistische Masszahlen dienen zur Beschreibung und zum Vergleich von Häufigkeitsverteilungen[13]. Die bekannteste und am häufigsten angewandte statistische Kenngrösse ist der Mittelwert; er charakterisiert eine Messreihe durch einen einzigen Wert.
Falls die n Einzelwerte einer Messreihe durch x_i ($i = 1, 2, \ldots, n$) bezeichnet werden, ist das arithmetische Mittel $\bar{x}$ gegeben durch

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

Bei Klassenbildung gilt entsprechend

$$\bar{x} = \frac{1}{n} \sum_{m=1}^{k} h_m u_m.$$

Dabei bedeuten h_m die absolute Häufigkeit der m-ten Klasse, u_m die entsprechende Klassenmitte, k die Anzahl der Klassen und n den Umfang der Messreihe, der sich auch aus der Summe der absoluten Häufigkeiten nach

$$n = \sum_{m=1}^{k} h_m$$

ergibt. Für Messreihen ohne Klassenbildung ist die absolute Häufigkeit immer gleich Eins.
Das arithmetische Mittel – entsprechendes gilt auch für die übrigen Kenngrössen einer Messreihe (Medien, Modus usw.) – vermag im allgemeinen zur Beschreibung von empirischen Verteilungen allein nicht zu genügen. Deshalb werden die Messreihen meist zusätzlich durch Streuungsmasse charakterisiert; sie orientieren über die Ausbreitung oder Streuung der Messwerte.
Als einfachstes Streuungsmass existiert die Spannbreite. R. Sie ist gleich der Differenz zwischen dem grössten und kleinsten Messwert der Messreihe.

13 Erwin Kreyszig. Statistische Methoden und ihre Anwendung. 3. Aufl., Göttingen 1968, S. 37–48; Regina Storm. Wahrscheinlichkeitsrechnung, mathematische Statistik und statistische Qualitätskontrolle. 3. Aufl., Leipzig 1969, S. 91–109.

$$R = x_{max} - x_{min}$$

Am häufigsten wird allerdings die Varianz als Streuungsmass verwendet. Sie wird mit s^2 bezeichnet und ist nach der Formel

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

zu berechnen. Die nichtnegative Quadratwurzel der Varianz heisst Standardabweichung oder mittlere quadratische Abweichung s.
Für gruppierte Messreihen errechnet sich die Varianz nach

$$s^2 = \frac{1}{n-1} \sum_{m=1}^{k} (u_m - \bar{x})^2 h_m.$$

Zur Relativierung der Standardabweichung s dient der Variationskoeffizient v. Er ist definiert durch

$$v = \frac{s}{\bar{x}} \cdot 100 (\%).$$

v drückt mithin die Standardabweichung s in Prozent des arithmetischen Mittels $\bar{x}$ der Messreihe aus und kann deshalb zum Vergleich mehrerer empirischer Verteilungen unter Berücksichtigung der üblichen Masszahlen herangezogen werden.
Um bei der Berechnung von Varianz und Standardabweichung eine vorausgehende Berechnung des Mittelwerts und damit auch in einem Programm die Speicherung der einzelnen Messwerte x_i zu vermeiden, werden häufig die numerisch stabileren Formeln

$$s^2 = \frac{1}{n(n-1)} \{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2\}$$

$$s^2 = \frac{1}{n(n-1)} \{n \sum_{m=1}^{k} h_m u_m^2 - (\sum_{m=1}^{k} h_m u_m)^2\}$$

verwendet und damit die Messwerte x_i und u_m direkt verarbeitet. Das folgende BASIC-Programm setzt keine geordneten Messreihen voraus.

```
    PRINT *STATISTISCHE MASSZAHLEN*
    PRINT *----------------------*
    REM INITIALISIERUNG
    LET Z=0
 10 LET D1=10**98
    LET D2=-D1
    LET X1=0
    LET X2=0
    LET N=0
```

```
   LET Z=Z+1
   REM DATENEINLESUNG UND VERARBEITUNG
   READ A$
   IF A$ = ≠ENDE≠ GOTO 50
   IF A$ = ≠MIT≠ GOTO 15
   LET F=1
15 READ X                          ↓  X = MESSWERT
   IF X$ = ≠STOP≠ GOTO 40
   IF A$ = ≠OHNE≠ GOTO 17
   READ F                          ↓ F = HAEUFIGKEIT
17 IF D1 < X GOTO 20
   LET D1=X
20 IF D2 GE X GOTO 30
   LET D2=X
30 LET X1=X1+X*F
   LET X2=X2+X**2*F
   LET N=N+F
   GOTO 15
40 REM GESAMTAUSWERTUNG
   PRINT
   PRINT ≠GESAMTAUSWERTUNG *≠;A$;≠* HAEUFIGKEITEN≠
   PRINT
   PRINT ≠       SERIE≠,≠    ANZAHL DER≠,≠       SPANN-≠,≠       MITTEL-≠,
   PRINT ≠      VARIANZ≠,≠       STANDARD-≠,≠   VARIATIONS-≠
   PRINT ≠ ≠,≠        WERTE≠,≠       WEITE≠,≠        WERT≠,≠ ≠,≠      ABWEICHUNG≠,
   PRINT ≠    KOEFFIZIENT≠
   PRINT
   PRINT Z,
   PRINT N,
   PRINT ABS(D1-D2),
   PRINT X1/N,
   PRINT (N*X2-X1**2)/(N*(N-1)),
   PRINT SQR((N*X2-X1**2)/(N*(N-1))),
   PRINT SQR((N*X2-X1**2)/(N*(N-1)))/(X1/N)*100.0
   PRINT
   GOTO 10
50 PRINT
   PRINT ≠--------ENDE---------≠
   END LIST DATA

LIST OF DATA AFTER END STATEMENT


≠MIT≠
10.15,1,10.25,2,9.85,3,10.10,1,9.70,2,9.95,3,10.05,4,10,5
≠STOP≠
≠OHNE≠
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17
≠STOP≠
≠ENDE≠
*

*** EVERYTHING SEEMS OK -- LET≠S GO AHEAD
```

STATISTISCHE MASSZAHLEN

GESAMTAUSWERTUNG *MIT * HAEUFIGKEITEN

SERIE	ANZAHL DER WERTE	SPANN-WEITE	MITTEL-WERT	VARIANZ	STANDARD-ABWEICHUNG	VARIATIONS-KOEFFIZIENT
1	21	0.55	9.9881	0.0209762	0.144832	1.45004

GESAMTAUSWERTUNG *OHNE * HAEUFIGKEITEN

SERIE	ANZAHL DER WERTE	SPANN-WEITE	MITTEL-WERT	VARIANZ	STANDARD-ABWEICHUNG	VARIATIONS-KOEFFIZIENT
2	17	16	9	25.5	5.04975	56.1084

--------ENDE---------

Wird zwischen zwei Zufallsvariablen X und Y, die in Form von Wertepaaren x_i und y_i vorliegen, ein linearer Zusammenhang von Y bezüglich X vermutet, so kann dies durch eine lineare Regressionsgerade ausgedrückt werden und deren Beschreibungsgüte durch eine Korrelationsrechnung nachgewiesen werden[14].
Ausgehend von der vermuteten Geradengleichung

$$y = bx + a$$

wird das Minimum der Summe der vertikalen Abstandsquadrate aller Messwerte von dieser Geraden aus der Funktion

$$k = \sum_{i=1}^{n} (y_i - bx_i - a)^2$$

gesucht. Setzt man die partiellen Ableitungen $\partial k/\partial a$ und $\partial k/\partial b$ dieser Funktion gleich Null, so erhält man daraus die Koeffizientengleichungen für a und b.

$$b = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

$$a = \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x^2 - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

Bei der Programmierung der Regressions- und Korrelationsrechnung bildet man zunächst die Summen

$$XS = \sum_{i=1}^{n} x_i, \quad XS2 = \sum_{i=1}^{n} x_i^2,$$

$$YS = \sum_{i=1}^{n} y_i, \quad YS2 = \sum_{i=1}^{n} y_i^2,$$

$$ZS = \sum_{i=1}^{n} x_i y_i$$

und verwendet diese zur Berechnung der Varianzen von X, Y und XY, wobei auf die im Abschnitt 431 erwähnten Gleichungen zurückgegriffen wird. Demzufolge gilt:

14 Kreyszig, Statistische Methoden und ihre Anwendung, S. 257–267 und S. 300–307; Müller-Merbach, Operations Research, S. 438–440.

$$s_x^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 = \frac{1}{n(n-1)} (n \cdot XS2 - (XS)^2)$$

$$s_y^2 = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y})^2 = \frac{1}{n(n-1)} (n \cdot YS2 - (YS)^2)$$

Analog definiert sich die Kovarianz s_{xy}

$$s_{xy} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})\,(y_i - \bar{y}) = \frac{1}{n(n-1)} (n \cdot ZS - XS \cdot YS).$$

Das Steigungsmass b der Geraden (Regressionskoeffizient von Y bezüglich X) ermittelt sich nun als Quotient der Kovarianz s_{xy} und der Varianz s_x^2 von X.

$$b = \frac{s_{xy}}{s_x^2}$$

Der Schnittpunkt a mit der Y-Achse bestimmt sich nach

$$a = \frac{1}{n} (YS - b \cdot XS).$$

Als Gütemass inwieweit Y und X zufällig voneinander abhängen, dienen der Korrelationskoeffizient r und das Bestimmtheitsmass b_{xy}, wobei dieses als Verhältnis der Streuung der aus der Regressionsgeraden ermittelten Funktionswerte zur Gesamtstreuung definiert ist[15].

$$b_{xy} = \frac{\frac{1}{n-1} \sum_{i=1}^{n} (bx_i + a - \bar{y})^2}{\frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y})^2}$$

Unter Verwendung der Varianzen ergibt sich

$$b_{xy} = \frac{s_{xy}^2}{s_x^2 s_y^2}.$$

15 Viktor Fetzer. Einführung in die Grundlagen der mathematischen Statistik. Heidelberg 1973, S. 137–142.

Der Korrelationskoeffizient r errechnet sich mit der Kovarianz s_{xy} und den Standardabweichungen von X und Y nach der Formel

$$r = \frac{s_{xy}}{s_x s_y}.$$

Je näher der absolute Wert von r an 1 liegt, um so besser wird die vermutete lineare Abhängigkeit zwischen X und Y nachgewiesen.
Die Streuung (Standardfehler) der Beobachtungswerte y_i um die Regressionsgerade berechnet sich aus der Varianz von Y und dem Bestimmtheitsmass nach der folgenden Gleichung[16]

$$s_f = \sqrt{s_y^2 (1 - b_{xy})}.$$

Das nachfolgende BASIC-Programm überprüft vor dem reihenweisen Einlesen der unabhängigen Werte x_i in den Vektor V und der abhängigen Werte y_i in den Vektor W – beide mit der Dimension N9 –, ob die Anzahl n der Wertepaare grösser als 2 und kleiner oder gleich N9 ist. Mit dem Schalter A1 kann die Ausgabe des Programms gesteuert werden. Für A1 ungleich Null wird zunächst eine Liste mit den Eingabedaten, x^2_i, y^2_i und $x_i y_i$ und anschliessend die Gesamtauswertung ausgegeben. Für A1 gleich Null unterbleibt der Ausdruck der Liste.

```
5     REM LINEARE REGRESSION UND KORRELATION
10    REM ***********************************************************************
15    REM ***** WENN DIE EINGELESENENEN DATA NICHT AUSGEDRUCKT WERDEN SOLLEN IST
16    REM ***** FUER A1 EINE 0 (NULL) EINZUGEBEN,SONST EINE ZAHL UNGLEICH NULL
20    REM ***********************************************************************
30    READ A1
40    REM ***********************************************************************
100   PRINT# LINEARE REGRESSIONS- UND KORRELATIONSANALYSE#
105   REM ***** DIE DIMENSIONIERUNG IST ABHAENGIG VON DER DATENMENGE *****
110   DIM V(100),W(100)
120   REM ***********************************************************************
130   PRINT
140   REM ***** ANZAHL DER ZAHLENPAARE WERDEN EIGELESEN *****
142   REM ***** N9 = GRENZE DER DIMENSIONIERUNG *****
145   LET N9=100
150   READ N
155   IF N <= N9 GOTO 160
157   PRINT #DIM FRWEITERN, DA V UND W NUR BIS#,N9,# DIMENSIONIERT#
158   GOTO 9999
160   IF N GE 3 THEN 200
170   PRINT# ----- FEHLER BEI EINGABE VON N, ES MUESSEN MEHR#,
171   PRINT#ALS ZWEI PAARE EINGEGEBEN WERDEN#
180   PRINT
190   GOTO 9999
200   REM ***** DIE X UND Y WERTE WERDEN REIHENWEISE EINGELESEN *****
203   REM X WIRD IN V EINGELESEN
206   REM Y WIRD IN W EINGELESEN
210   FOR I=1 TO N
220     READ V(I)
230   NEXT I
231   FOR I=1 TO N
232     READ W(I)
234   NEXT I
240   REM ***** SUMMENBILDUNG *****
250   FOR I=1 TO N
```

16 Hans Bartel. Statistik I. 2. Aufl., Stuttgart 1974, S. 115–116.

```
260     LET X=X+V(I)
270     LET Y=Y+W(I)
280     LET X2=X2+V(I)**2
290     LET Y2=Y2+W(I)**2
300     LET Z=Z+V(I)*W(I)
310   NEXT I
350   REM ***** BERECHNUNG DER VARIANZEN *****
355   LET X3=(N*X2-X**2)/(N*(N-1))
360   LET Y3=(N*Y2-Y**2)/(N*(N-1))
365   LET K3=(N*Z-X*Y)/(N*(N-1))
370   REM ***** BERECHNUNG STEIGUMG UND SCHNITTPUNKT MIT Y-ACHSE *****
375   LET B=K3/X3
380   LET A=(Y-B*X)/N
385   REM ***** BERECHNUNG DES BESTIMMTHEITSMASSES *****
390   LET B1=K3**2/(X3*Y3)
400   REM *****BERECHNUNG KORRELATIONSKOEFFIZIENT *****
410   LET R=K3/(SQR(X3)*SQR(Y3))
420   REM ***** BERECHNUNG DES STANDARDFEHLERS *****
430   LET S1=SQR(Y3*(1-B1))
440   PRINT
450   PRINT
455   REM ********* AUSDRUCK DER ERGEBNISSE *********
460   PRINT# ---- ERGEBNISSE DER LINEAREN REGRESSIONS-#!
470   PRINT# UND KORRELATIONSANALYSE ----#
480   PRINT
485   PRINT
487   IF A1=0 THEN  536
490   PRINT#  X-VARIABLE#,#  Y-VARIABLE#,#  X**2 WERTE#,#  Y**2 WERTE#,
491   PRINT#   X*Y WERTE#
496   PRINT
500   FOR I=1 TO N1
511     PRINT V(I),W(I),V(I)**2,W(I)**2,V(I)*W(I)
520   NEXT I
535   PAGE
536   PRINT
539   PRINT#+-----------------------------------------+#
540   PRINT#I                                         I#
541   PRINT#I  ANZAHL DER WERTEPAARE.....#,N,#I#
550   PRINT#I                                         I#
560   PRINT#I  SUMME DER X-WERTE.........#,X,#I#
570   PRINT#I  SUMME DER Y-WERTE.........#,Y,#I#
580   PRINT#I  SUMME VON X*Y.............#,Z,#I#
590   PRINT#I  SUMME DER X-QUADRATE......#,X2,#I#
600   PRINT#I  SUMME DER Y-QUADRATE......#,Y2,#I#
610   PRINT#I  MITTELWERT VON X..........#,X/N,#I#
620   PRINT#I  MITTELWERT VON Y..........#,Y/N,#I#
622   PRINT#I  VARIANZ VON X.............#,X3,#I#
623   PRINT#I  VARIANZ VON Y.............#,Y3,#I#
624   PRINT#I  KOVARIANZ.................#,K3,#I#
630   PRINT#I  STEIGUNG B DER GERADEN....#,B,#I#
640   PRINT#I  SCHNITTP. MIT DER Y-ACHSE.#,A,#I#
650   PRINT#I                                         I#
660   PRINT#I  STANDARDFEHLER............#,S1,#I#
670   PRINT#I                                         I#
680   PRINT#I  KORRELATIONSKOEFFIZIENT...#,R,#I#
681   PRINT#I                                         I#
682   PRINT#I  BESTIMMTHEITSMASS.........#,B1,#I#
683   PRINT#I                                         I#
684   PRINT#+-----------------------------------------+#
999   DATA 0
1000  DATA 20
1010  DATA 167,178,158,181,174,195,168,173,178,160
1011  DATA 169,188,153,176,177,161,193,201,175,156
1020  DATA 60,68,70,92,88,104,68,73,79,62
1021  DATA 72,89,53,78,79,67,110,121,92,63
9999  END

*** EVERYTHING SEEMS OK -- LET#S GO AHEAD

   PERCENT OF AVAILABLE STORAGE USED                45.135
   PERCENT OF AVAILABLE DATA STORAGE USED           12.727
   PERCENT OF AVAILABLE NUMBERED STATEMENTS USED    29.706
```

LINEARE REGRESSIONS- UND KORRELATIONSANALYSE

---- ERGEBNISSE DER LINEAREN REGRESSIONS- UND KORRELATIONSANALYSE ----

ANZAHL DER WERTEPAARE.....	20
SUMME DER X-WERTE.....	3481
SUMME DER Y-WERTE.....	1588
SUMME VON X*Y.....	280449
SUMME DER X-QUADRATE.....	609187
SUMME DER Y-QUADRATE.....	132088
MITTELWERT VON X.....	174.05
MITTELWERT VON Y.....	79.4
VARIANZ VON X.....	174.682
VARIANZ VON Y.....	315.832
KOVARIANZ.....	213.558
STEIGUNG B DER GERADEN....	1.22256
SCHNITTP. MIT DER Y-ACHSE.	-133.386
STANDARDFEHLER.....	7.399
KORRELATIONSKOEFFIZIENT...	0.90921
BESTIMMTHEITSMASS.....	0.826663

BASIC

Von Prof. Dr. Karl Weber und Dipl.-Ing. C. W. Türschmann sind erschienen:

als Fortsetzung des vorliegenden Bandes

BASIC.
Lehr- und Handbuch der Programmiersprache BASIC mit wirtschaftswissenschaftlichen Anwendungsbeispielen.

Band 2: Informationen über Maschinensysteme – BASIC-Problembearbeitung und -Sonderanweisungen – BASIC-Beispielprogramme – BASIC-Literatur
"Uni-Taschenbücher" 589. 244 Seiten, kartoniert DM 20.80

als weiterführende Begleitstoffe

FOSBIC-Compiler
Ein BASIC-Compilersystem auf FORTRAN-Basis.
224 Seiten mit vielen Zeichnungen und Compiler-Ausdrücken. Kartoniert Fr. 40.– / DM 44.–

BASIC Programmier-Richtlinien für Austauschprogramme

81 Seiten, kartoniert Fr. 30.– / DM 33.–

VERLAG PAUL HAUPT BERN UND STUTTGART

Die Grundlagen

Prof. Dr. Karl Weber

BASIC für Anfänger

Hier legt der Autor Wert auf einfache Darstellung der BASIC-Sprachstruktur und auf die Vermittlung leicht verständlicher Beispiele.

153 Seiten mit 17 Abbildungen und 4 Tabellen, kartoniert Fr. 28.–/DM 31.–

IDV AG für EDV-Ausbildung

EDV in 20 Stunden

Die wichtigsten Grundlagen. Ein Lernprogramm in 46 Lernschritten mit 82 Fotos und Illustrationen, 81 Fragen und Lösungen.

7. Auflage, 104 Seiten, kartoniert Fr. 18.– / DM 19.80

Prof. Dr. Hansjürg Mey (Herausgeber)

Mikroelektronik

Chancen und Gefahren für die schweizerische Wirtschaft

72 Seiten mit 10 graphischen Darstellungen, kartoniert Fr. 14.80/DM 16.50

VERLAG PAUL HAUPT BERN UND STUTTGART

Das vorliegende Lehr- und Handbuch soll als BASIC-Einführung und zur BASIC-Vertiefung dienen.

Es orientiert in einem ersten Kapitel über die BASIC-Entwicklung im allgemeinen und vermittelt einen Überblick über eine Reihe von Maschinensystemen, die BASIC als Programmiersprache zulassen. Das zweite Kapitel führt in die Grundlagen der BASIC-Programmierung ein. Im dritten Kapitel wird BASIC unter besonderer Berücksichtigung der BASIC-Elementaranweisungen behandelt. Der metalinguistischen Darstellungsweise von Backus folgend, wird die BASIC-Syntax dargestellt und verbal die BASIC-Semantik erörtert. Zur Illustration der theoretischen Ausführungen dienen zahlreiche Beispiele, die mit dem FOSBIC-Compiler bearbeitet wurden. Das vierte Kapitel vermittelt einige praxisnahe, speziell für Wirtschaftswissenschaftler relevante Beispielprogramme.

UTB Wirtschaftswissenschaften
Informatik / Datenverarbeitung

DM 23.80 ISBN 3-258-03046-1